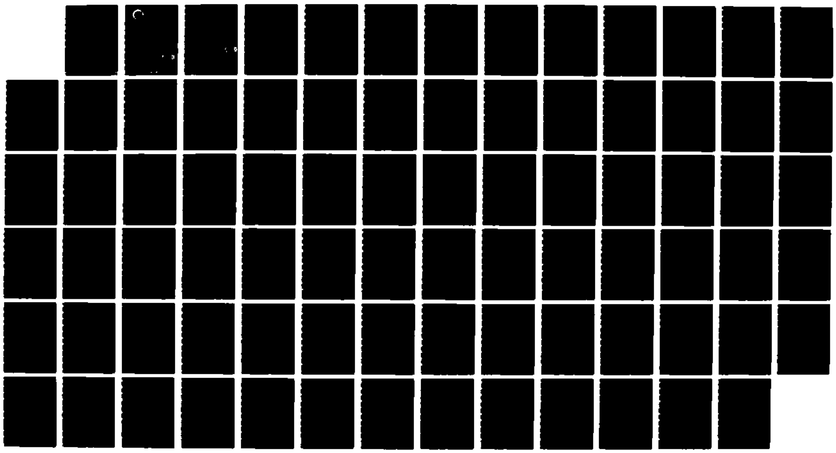
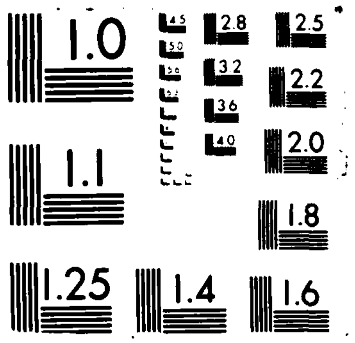
AD-A181 431 HEURISTIC PROCEDURES FOR 0-1 INTEGER PROGRAMMING(U) 1/1
STANFORD UNIV CA SYSTEMS OPTIMIZATION LAB
K A ERCIKAN ET AL MAR 87 SOL-87-3 N00014-85-K-0343
UNCLASSIFIED F/G 12/4 NL




1.0
4.5
5.0
5.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

2

Systems
Optimization
Laboratory

HEURISTIC PROCEDURES FOR 0 – 1 INTEGER PROGRAMMING

by
Kadriye A. Ercikan and
and
Frederick S. Hillier

TECHNICAL REPORT SOL 87-3
March 1987

Department of Operations Research
Stanford University
Stanford, CA 94305

87 6 4 091

2

SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305-4022

HEURISTIC PROCEDURES FOR 0 – 1 INTEGER PROGRAMMING

by
Kadriye A. Ercikan and
and
Frederick S. Hillier

TECHNICAL REPORT SOL 87-3
March 1987

Research and reproduction of this report were partially supported by the Office of Naval Research Contract N00014-85-K-0343.

TABLE OF CONTENTS

LIST OF TABLES

Chapter 1

Introduction

1.1. Formulation

Many decision making problems can be formulated as a 0-1 integer program. The computation time for the existing algorithms for solving these problems increases rapidly with the size of the problem. Even with today's computers, sometimes it is not possible to obtain optimal solutions for these problems. Therefore, heuristic procedures can either be used to find a good approximate solution to the problem or to increase the efficiency of an optimal algorithm by obtaining a good starting solution.

This thesis presents heuristic procedures for 0-1 linear programming problems. These are based on Hillier's heuristic procedures for pure integer linear programming [7,16,18]. The original procedures when tested were consistently close to optimal and frequently had actually been optimal. They were designed for general integer programming problems. Therefore, they were mainly tested on such problems. The aim in this thesis has been to streamline these procedures to exploit the structure of 0-1 integer programming. The procedures were designed for the following pure 0-1 integer programming problem.

$$\text{maximize } Z = \sum_{j=1}^{n} c_j x_j ,$$

subject to

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i \qquad (i = 1,2, . . ,m) \qquad (1)$$

$$x_j \geq 0 \qquad (j = 1,2, . . ,n) \qquad (2)$$

$$x_j = 0 \text{ or } 1 \qquad (j = 1,2, . . ,n) \qquad (3)$$

Three main procedures have been studied. Some of these procedures assume some of the following:

$$c_j \geq 0 \qquad (j = 1,2, . . ,n) \qquad (4)$$

$$a_{ij} \geq 0 \qquad \begin{matrix}(i = 1,2, . . ,m)\\(j = 1,2, . . ,n)\end{matrix} \qquad (5)$$

$$b_i \geq 0 \qquad (i = 1, . . ,m) \qquad (6)$$

$$c_j \text{ is an integer} \qquad (j = 1,2, . . ,n) \qquad (7)$$

Procedure 3 assumes all four. Therefore, it is designed for multi-constraint knapsack type problems. Procedure 2 assumes (4), (6) and (7). However, since (5) is not assumed, a problem with negative objective coefficients can easily be transformed into the required form by substituting $(1 - x_j^{'})$ throughout the model (where $x_j^{'}$ also is a binary variable) for each x_j with $c_j < 0$. Procedure 1 assumes only (7) and that the set of solutions that satisfy constraints (1) and (2) possesses an interior point. Note that any objective function with rational coefficients can be transformed to satisfy (7) bv multiplying through by a common denominator.

The notation used throughout this thesis is consistent with [16,18]. For Procedure 1 and parts of Procedures 2 and 3, the constraints are normalized so that they become:

$$\sum_{j=1}^{n} a'_{ij}x_j < b'_i$$

where

$$a'_{ij} = a_{ij} \Big/ \sqrt{\sum_{j=1}^{n} a^2_{ij}} \qquad \begin{array}{l}(i = 1,2, . . ,m)\\(j = 1,2, . . ,n)\end{array}$$

$$b'_i = b_i \Big/ \sqrt{\sum_{j=1}^{n} a^2_{ij}} \qquad (i = 1,2, . . ,m).$$

b' is the Euclidean distance from the hyperplane, $\sum_{j=1}^{n} a_{ij}x_j = b_i$, to the origin.

1.2 Survey of Related Work

Over the past 30 years, there has been substantial research on developing algorithms for finding an optimal solution for integer programming problems. In [9] these algorithms are grouped according to whether they are based primarily on enumeration, Bender's Decomposition, cutting planes, or group theory. Enumerative algorithms include those which use implicit enumeration and branch-and-bound. For the pure integer programming problem, enumerative algorithms have been developed by Balas [1], Hillier [17], Faaland and Hillier [7], Geoffrion [8], Glover [10], Hammer and Rudeam [15], Lemke and Spielberg [22], and Woiler [33], among others. The above algorithms base their fathoming tests mainly on the logical implications of the problem constraints. The first branch-and-bound algorithm, which was developed by Land and Doig [21] for mixed as well as pure integer programs, bases its fathoming test mainly on associated linear programs. An improved variation of this algorithm subsequently was developed by Dakin [5]. Bender's approach [3] is used for mixed integer programming, since it essentially decomposes a mixed problem down to solving an alternating sequence of pure integer and pure linear problems. The cutting-plane approach was the first general approach taken to solving integer programs. The foundations of this approach were laid by Gomory [11,12]. His algorithms deal with dual feasible solutions, so that a primal feasible all-integer solution is not obtained until an optimal solution is reached. The Group Theoretic approach also was intiated by Gomory [13]. Further studies of this type have been done by Shapiro [27,28,29], Glover [10], Thiriez [30], and Wolsey [34]. This approach is generally applied to pure integer problems. A more recent algorithm by Crowder et al. [4], uses a combination of problem preprocessing, cutting planes, and the branch-and-bound technique. Their computational experience on large scale pure zero-one linear problems has been impressive.

Because of the significant computational limitations of integer programming algorithms for obtaining an optimal solution, there has been considerable research on heuristic algorithms for efficiently seeking very good solutions that are not guaranteed to be optimal. Such algorithms have been developed by Balas and Martin [2], Reiter and Rice [23], Echols and Cooper [6], Senju and Toyoda [26], Hillier [16,18], Faaland and Hillier [7], Roth [25], Kochenberger, McCard and Wyman [20], Ibaraki, Ohashi, and Mine [19], and Toyoda [30]. The ones presented in [2], [26] and [31] are specifically designed for the binary integer programming case.

Balas and Martin [2] use the fact that a 0-1 program is equivalent to the associated linear program with the added requirement that all slack variables, other than those in the upper bound constraints, be basic. Toyoda [31] assigns measures of preferability to zero-one variables that change the values of the variables from zero to one. Senju and Toyoda [26] start the heuristic search from an initial solution which has all $x_j = 1$, and then the variables that provide the smallest contribution to objective function increase per unit of weighted infeasibility are dropped to zero.

Since the heuristic procedures developed in this thesis for 0-1 integer programming are based on Hillier's procedures for general integer programming, Hillier's procedures are described in some detail in the next chapter under the label of "Original Procedures."

Zanakis [35] examined the performance of three heuristic methods (Senju-Toyoda [26], Kochenberger et al. [20], and Hillier [16]) when applied to the 0-1 linear programming problem with nonnegative coefficients.

Since the latter two algorithms were designed for general integer linear programming, Zanakis simply added upper bounds of one on the variables without any streamlining for this special structure (not even the upper bound technique for the simplex method). The effectiveness of each algorithm was measured in terms of computing time, error and relative error. According to the test results, Hillier's algorithm was the most accurate but not as fast as the other two. Kochenberger's et al. heuristic was the fastest of the three in tightly constrained problems. In general, the Senju-Toyoda algorithm tended to be the fastest, but was the least accurate on small and medium size problems.

The heuristic algorithms developed here are designed so that they will be as accurate as Hillier's original algorithms without requiring as much computational effort because they are designed specifically for the 0-1 integer programming case.

Chapter 2

Construction of the Procedures

In constructing Procedures 1,2 and 3, the aim has been to decrease the computation time for Hillier's pure Integer Programming Heuristic Procedures by considering that the values of the variables can only be 0 or 1. For Procedures 2 and 3, the additional special structure assumed also is considered.

The original procedures have a three-phase approach. Phase 1 identifies a general region within which to explore for good feasible solutions by finding the optimal non-integer solution by the simplex method and a second point well into the feasible region. Phase 2 searches for a feasible integer solution by moving along the line segment from the first point to the second to initiate searches. Phase 3 tries to improve on the feasible solution obtained in Phase 2. The final solution in this phase is the desired approximate solution.

In the present procedures, certain changes have been made in different phases. In the original procedures, alternative methods were introduced for each phase. After examining the test results of the original procedure [16,18], the apparent best method for each phase has been selected. In some cases, phases have been changed completely in order to find a more appropriate method for the 0-1 integer programming case. Each procedure will be described in detail in the following sections.

2.1 Procedure 1

Procedure 1 is based directly on the heuristic procedures for general ILP in [16]. Therefore, it also has three phases. Certain changes and streamlining have been incorporated into each phase. The following subsections give a summary description of each phase of the original procedures followed by a discussion of the changes and streamlining for the 0-1 case.

a. Phase 1

(i) Original Procedures

Phase 1 of this procedure starts by solving the LP-relaxation of the problem to find its optimal solution $x^{(1)}$. The next step is to find a second point $x^{(2)}$ well into the feasible region. Phase 1 ends by constructing the line segment between the two points. [16] provides two methods (labeled 1 and 2) for finding $x^{(2)}$, [7] generalizes the approach to finding a piecewise linear path, and [18] provides another generalization.

(ii) Changes for the 0-1 Case

For the first step, the simplex method with the upper bound technique is used to find $x^{(1)}$. Methods 1 and 2 of the original procedures do not require that either $x^{(2)}$ or the corresponding rounded solution satisfy all of the constraints (2) and (3) that are not binding at $x^{(1)}$. Therefore, an interior path found by considering all the constraints rather than only those that are binding at $x^{(1)}$ should be more effective in Phase 2. The following two methods drawn from [7] give piecewise linear interior paths.

The first method, which will be denoted as 1a, generates the piecewise linear path by obtaining the parametric solution to the linear program:

$$\max r,$$

subject to:

$$\sum_{j=1}^{n} a_{ij}x_j + \Delta_i r \leq b_i \qquad (i = 1,2, . . ,m)$$

$$\sum_{j=1}^{n} c_j x_j = Z$$

$$x_j \leq 1$$

$$x_j \geq 0 \qquad (j = 1,2, . . ,n)$$

$$r \geq 0$$

as Z is decreased from its value at $x^{(1)}$, and then deleting r from the parametric solution. This method stops when max r reaches its largest value, and the corresponding solution for x is $x^{(2)}$.

The second method, 2a, obtains the breakpoints of the piecewise linear path as the basic feasible solutions (after deleting r) generated in the process of solving the following problem:

$$\max r,$$

subject to:

$$\sum_{j=1}^{n} a_{ij}x_j + \Delta_i r \leq b_i \qquad (i = 1,2, . . ,m)$$

$$\begin{aligned} x_j &\leq 1 \\ & \qquad\qquad (j = 1,2, . . ,n) \\ x_j &\geq 0 \end{aligned}$$

$$r \geq 0$$

starting with the initial solution $x^{(1)}$. The solution for x that maximizes r is $x^{(2)}$.

For either method, Δ_i can be one of the following:

$$\Delta_i = 1/2 \sum_{j \in B} |a_{ij}| \qquad (i)$$

$$\Delta_i = 1/2 \left(\sum_{j=1}^{n} a_{ij}^2 \right)^{1/2} N^{1/2} \qquad (ii)$$

$$\Delta_i = \left(\sum_{j=1}^{N} a_{ij}^2 \right)^{1/2} \qquad (iii)$$

where B is the set of basic variables from among $\{x_1, x_2, . . , x_n\}$ in $x^{(1)}$ and N is the number of elements in B.

An alternative to Methods 1a and 2a would be to use the linear path between $x^{(1)}$ and $x^{(2)}$ instead of the piecewise linear path for initiating the search for a feasible solution. Since both methods obtain the same $x^{(2)}$, the quicker Method 2a should be used. Using Method 2a to obtain $x^{(2)}$ and then simply constructing the linear path between $x^{(1)}$ and $x^{(2)}$ is labeled as Method 2b.

Method la requires a software package that includes parametric programming as well as a considerable amount of execution time. The available computer package for this study, Lindo, did not include parametric programming. Considering that Methods 2a and 2b require less time, they were chosen for Procedure 1. Test results in [7] show that the first definition of Δ_i should be preferred to the second one. Therefore the first and third definitions are used. When Method 2a is used, the sequence of basic feasible solutions generated is recorded and each successive pair is connected by a line segment to form a piecewise linear path. For Method 2b, only the line segment joining $x^{(1)}$ and $x^{(2)}$ is used for the search. This completes

Phase 1.

b. <u>Phase 2</u>

(i) Original Procedures

The aim of this phase is to find a feasible 0-1 solution between the two points, $x^{(1)}$ and $x^{(2)}$, found in Phase 1. Method 1 for Phase 2 consists of moving continuously down the line segment from $x^{(1)}$ to $x^{(2)}$, rounding to the nearest integer solution, until the rounded solution is feasible. Any point on the line segment can be represented as:

$$x = (1-\alpha)\ x' + \alpha\ x''$$

where $0 \leq \alpha \leq 1$. α is first set to 0; if the solution obtained by rounding x is feasible, then Phase 2 terminates with this as the desired feasible solution. If the solution obtained by rounding x is not feasible, then α is increased to the next such that the resulting x obtained would give a different rounded solution. Phase 2 ends when α is greater than 1 or a feasible solution is obtained.

Method 2 differs from Method 1 in that α is increased by fixed amounts and each time the nearby region is searched for a feasible 0-1 solution. For each value of α, the first step is to apply scientific rounding to the components of x in order to identify the nearest integer solution. If the rounded solution is not feasible, then check to see if increasing or decreasing any variable by one will decrease the "infeasibility" q. If there are no such variables, then go to the next value of α. If there is exactly one such variable, then make this change. If there is more than one variable that can be changed to decrease the infeasibility q, then select the one which will give the largest "improvement" p.

Using the notation, $(y)_+ = \max\{0,y\}$, two alternative definitions of the "infeasibility" q are the following:

$$\text{(i)} \quad q = \sum_{i=1}^{m} \Big(\sum_{j=1}^{n} a'_{ij}x_j - b'_i \Big)_+ ,$$

which is the sum of the Euclidean distances between x and each of the violated constraining hyperplanes;

$$\text{(ii)} \quad q = \max_{i\varepsilon\{1,\,.\,.\,,m\}} \Big\{ \sum_{j=1}^{n} a'_{ij}x_j - b'_i \Big\},$$

which is the maximum of the Euclidean distances between x and the violated constraining hyperplanes.

Three alternative definitions of the "improvement" p are the following:

(i) $p = -\Delta q$,

where Δq is the change in q resulting from the change in the variable x_j;

(ii) $p = c_j \Delta x_j \,/\, (-\Delta q)$,

where Δx_j is the change in x_j being made;

(iii) $p = -\Delta q + c_j' \Delta x_j$

where c_j' is the normalized value of c_j.

The first definition of p is a natural measure for the "improvement" in infeasibility obtained by changing the value of a variable x_j, but it does not take into account the change in the value of the objective function. The second definition of p does take this into account by selecting the change that increases the objective function the most per unit decrease in q. Therefore, when the feasible solution is reached, the objective function value will tend to be relatively large. The third definition is similar to the first one except for an added term that also considers the effect on the objective function. This definition encourages large moves toward the most attractive portion of the feasible region.

With alternative definitions of p and q, different criteria can be found for choosing the variable to be changed. Using the notation in [16,18], some of these criteria are as follows:

Criterion A:	first definiton of p, first definition of q
Criterion B:	first definition of p, second definition of q
Criterion C:	second definition of p, first definition of q
Criterion D:	second definition of p, second definition of q
Criterion E:	third definition of p, first definition of q
Criterion S:	first definition of q. This is a streamlined approach. As soon as a possible change that yields an improvement is found, it is implemented without finding and comparing all the other improving changes.

Criteria A and B are based on the measurement of the infeasibility and they do not consider the change in the objective function. When the original procedures were tested in [18] the results showed that Criterion A was generally better than B. Since these two criteria differ only in their definition of q, this suggests that the first definition of q is superior to the second. For this reason, Criterion C should be preferred to D. Further testing with the original procedures [18] has been done to try to distinguish between the four remaining criteria, A, C, E and S. However, the main conclusion is that even though large differences can occur on individual problems, the choice of a particular criterion does not have a strong effect on the average performance of the heuristic procedure in the long run.

Method 3 is a combination of Methods 1 and 2. As in Method 1, α is increased at each iteration by the minumum amount required to obtain a different rounded solution. However, rather than only checking this rounded solution for feasibility, the nearby region is also explored as in Method 2.

(ii) Changes for the 0-1 Case

In the present procedure, Method 3 with Criterion A has been used to find a feasible 0-1 solution between the two points, $x^{(1)}$ and $x^{(2)}$, found in Phase 1. In this case, the components of $x^{(1)}$ and $x^{(2)}$ are between 0 and 1 and the entire path between them generated by Method 2a or 2b of Phase 1 also has this property, so every rounded solution along this path is a 0-1 solution. If Method 2a had been used in Phase 1, the first iteration for Phase 2 starts with x' as $x^{(1)}$ and x'' as the first basic feasible solution in finding $x^{(2)}$. The search is initiated from the line segment between these two points. If a feasible solution is found, Phase 2 ends, but if a feasible solution is not found, the search is continued from the next line segment, which is the line joining the first and second basic feasible solutions obtained in finding $x^{(2)}$. If no feasible solution is found on this line segment move to the next one, etc., until a feasible solution is found. Method 2b of Phase 1 yields just a single line segment for Phase 2. Certain adjustments have been made for the 0-1 case in different steps of Method 3. These are as follows. Every integer solution considered now is required to be binary. Therefore, when Step 6 of Methods 2 and 3 in [16] determines in which direction each variable should be changed in order to decrease the infeasibility, the change would be considered now only if it would result in a 0-1 solution.

Phase 2 ends as soon as a feasible solution is found. There is no guarantee, in general, that this will occur.

c. <u>Phase 3</u>

(i) Original Procedures

Phase 3 starts with the feasible solution found in Phase 2 and then tries to improve on it. This was initially done by alternating two modes. The first mode tries to increase the objective function value by increasing or decreasing the value of a single variable by one, at the same time keeping the solution feasible. Two alternative methods are considered for this mode. When determining how much each variable can be changed in the favorable direction, Method 1 imposes integer restrictions on these quantities, whereas Method 2 does not. Test results in [18] suggested that Method 1 is better than Method 2. Therefore, since its relative appeal is even stronger in the 0-1 integer programming case, it was chosen for the present procedures.

(ii) a. Changes for the 0-1 Case in the First Mode

In Step 1 of Part II in [16], $d_{ij} = s_i / |a_{ij}|$ where s_i is the slack for constraint i. For the 0-1 case, d_{ij} is set to 0 when $c_j > 0$ and $x_j = 1$.

The second mode tries to obtain better feasible solutions by changing two variables simultaneously.

(ii) b. Changes for the 0-1 Case in the Second Mode

In Step 1 of Part IV, in addition to checking the sign of $x_j + \delta_j$, a check is made whether $x_j + \delta_j \le 1$ before permitting the change δ_j. The other change in this part is that once a change is made on a variable in the favorable direction, it is never considered again, i.e., the loop which goes back to Step 1 from Step 3 is removed. In Step 6 of Part V and Part VI, U_k is set to 1. In Part VII, after once considering a variable for change in the direction which would decrease the objective function value, it is never considered again, i.e., the loop which goes back to Step 1 from Step 5 is removed.

The two modes above are applied alternately until no better solution is found. This approach constitutes the first part of the method that has been used in the present procedures.

(iii) Other Methods from Original Procedures

Three other methods have been considered, namely, Methods 3, 4 and 5 from [18]. Method 3 starts with just the first mode of search described above before undertaking a new mode of search. Methods 4 and 5 complete Method 1 of Phase 3 (both modes of search) before they start the additional search for further improvement. In these methods, the new modes of search involve changing many variables in order to reach a better solution. It is computationally infeasible for large problems to consider all ways of changing several variables simultaneously. Therefore, methods that will efficiently consider only promising ways of changing many variables are needed. Let $x^{(L)}$ denote the current best feasible solution and $z^{(L)}$ its objective function value. All three methods are initiated by adding a new constraint, $cx \geq b_o$ where $b_o = z^{(L)} + 1$, to the problem. This makes $x^{(L)}$ infeasible and reduces the feasible region so that it only contains better feasible solutions. In all of the methods, one begins by moving from $x^{(L)}$ through a sequence of infeasible points that try to progress to a better feasible solution.

Methods 3 and 4 go through n cycles, in the general integer programming case, where each one begins by changing one of the n variables in the favorable direction. The first step in each cycle gives a new solution which is not feasible. Then a procedure similar to the one in Phase 2 is repeated. In other words, one tries to decrease the "infeasibility", q, by making changes in the variable which will give the best "improvement" p.

Method 5 is similar to Method 4, but instead of n cycles, there is only one. It starts with $x^{(L)}$, which now is infeasible because of the new constraint, $cx \geq b_o$. It then follows a procedure similar to the one in Phase 2 for finding a feasible solution. As adapted here, each iteration consists of finding which variable would give the largest "improvement" p according to Criterion A if the variable were changed to its other binary value, and then making that change.

Sometimes, largest p might be negative so this change will increase the infeasibility. Thus it might be necessary to move away from the feasible region initially, in order to be able to eventually find a better feasible solution. It is possible that a feasible solution is never reached. Therefore, to avoid moving away from the feasible region indefinitely, an upper limit, 100 is imposed on the number of iterations.

Both Method 3 and Method 4 require more than some multiple of mn^2 elementary operations, so that the running time grows rapidly with the size of the problem. Furthermore, previous testing [18] suggests that Method 5 tends to do beter than Methods 3 and 4 in reaching a better feasible solution that requires changing many variables, apparently because of its drifting ability.

(iv) Changes for the 0-1 Case

Method 5 has been chosen for the present procedures. The only change from the description in [18] is that the only trial solutions considered now are 0-1 solutions.

2.2 Procedure 2

This procedure assumes (4), (6) and (7). It starts with all the variables at 0, which is a feasible solution for (1) - (3). It then tries to raise the most promising variables to 1. This is done by finding how much each variable can be increased before it becomes infeasible according to (1). In particular, let

$$K_{ij} = \begin{cases} b_i / a_{ij}, & \text{if } a_{ij} > 0 \\ +\infty, & \text{if } a_{ij} \leq 0 \end{cases},$$

for $i = 1,2,\ldots,m$ and $j = 1,2,\ldots,n$,

and

$$R_j = \min_{i=1,2,\ldots m} K_{ij} \quad , \quad \text{for } j = 1,2,\ldots,n.$$

Then R_j indicates how much the variable x_j can be increased before violating (1). Now let

$$\text{Range}\ (x_j) = [R_j] \equiv (\text{greatest integer} \leq R_j) \quad ,$$
$$\text{for } j = 1,2,\ldots,n.$$

If there are k or more variables with Range $\geq k$, then this means that k of these variables can be set to 1 while retaining feasibility. Because of (4), increasing any variable x_j to 1 can only increase $z (c_j > 0)$ or leave it unchanged $(c_j = 0)$.

Each iteration begins by finding the largest integer k such that at least k variables have its Range $\geq$ k. If there are exactly k variables with Range $\geq$ k, then set all of them to 1. If there are more than k such variables, then set the k such variables with the highest objective row coefficients to 1.

After setting k variables to 1, the right hand side is adjusted in the following way. Let D be the set of indices of the k variables which were just set to 1. Reset

$$b_i = b_i - \sum_{j \varepsilon D}^{n} a_{ij} x_j \qquad \text{for } i = 1,2, \ldots, m.$$

New values are found for R and Range with the adjusted b_i's. The same procedure is repeated except for the variables which are already at 1. These variables are not considered again. This part of the procedure ends when Range is equal to 0 for all the variables. The above process can be summarized as follows:

1. Set $E = \emptyset$.
2. Calculate K_{ij} for $i = 1,2, \ldots, m$ and $j = 1,2, \ldots, n$.
3. Calculate R_j for $j = 1,2, \ldots, n$.
4. Calculate $\text{Range}(x_j)$ for $j = 1,2, \ldots, n$.
5. Determine the largest integer k such that there are k or more variables with Range $\geq$ k, and add the variables with Range $\geq$ k to the set E.
6. If k = 0, then go to step 8. Otherwise, if E has exactly k elements, then set all of them to 1; if E has more than k elements, then just set k variables in E with the highest objective row coefficients to 1.

7. Adjust the right hand side and return to step 2.

8. Stop.

The above process constitutes the first part of this procedure. The second part starts with the feasible solution obtained from the first part. It then tries to improve on it. Method 5 of Procedure 1 is used here. Before starting Method 5, the problem is normalized. Therefore, Procedure 2 differs from Procedure 1 in that Phases 1 and 2 of Procedure 1 is replaced by the first part of Procedure 2 for finding an initial good feasible solution.

2.3 Procedure 3

Procedure 3 is similar to Procedure 2 in that it tries to find a feasible solution in the first part and then adopts Method 5 of Procedure 1 to find a better feasible solution in the second part. Both procedures assume that $b_i > 0$ for $i = 1, \ldots, m$ and $c_j > 0$ for $j = 1,2, \ldots, n$, whereas Procedure 3 also assumes that $a_{ij} > 0$ for $i = 1,2, \ldots, m$ and $j = 1,2, \ldots, n$. The first part of Procedure 3 also starts with all variables at 0. The most promising variables to be set to 1 are found in a slightly different manner. R is found in the same way as before. Now a new quantity

$$P_j = c_j R_j$$

is calculated for each variable. This is a measure of how "profitable" (increase in the objective function) each variable can be if it alone were to be increased as much as (1) permits. In actuality, any variable that is increased would be increased to 1. It is desirable to choose the variables to be increased in a way that will allow further improvements. Therefore, it is necessary to consider the coefficients of each variable in the functional constraints (1). Choosing a variable to increase that has a relatively small sum of these coefficients should tend to leave relatively good opportunities for further improvements by then increasing other variables. Let

$$A_j = \sum_{i=1}^{m} a_{ij}, \qquad \text{for } j = 1,2, \ldots, n.$$

(If the coefficients of variables in different constraints differ significantly, then (1) needs to be normalized as shown at the end of Section 1.1 in order for A to make sense in the rest of the procedure.) The measure which determines which variable to set to 1 is

$$\text{Ratio}(x_j) = P_j \;/\; A_j, \text{ for } j = 1,2, . . ,n.$$

It is desirable that P_j be as high as possible and A_j as low as possible. When A_j is 0, set Ratio $(x_j) = +\infty$. If P_j is 0, then set Ratio $(x_j) = 0$. The variable maximizing Ratio is then set to 1. This completes one iteration. To start the next iteration, the right hand side is adjusted by resetting

$$b_i = b_i - a_{ij}x_j, \quad \text{for } i = 1,2, . . ,m \text{ and } j = 1,2, . . ,n,$$

for purposes of recalculating the R_j. Once a variable is set to 1, it is never considered again and so is never changed to 0 during this part of Procedure 3. The iterations for this part end when none of the remaining variables can be increased to one while retaining feasibility. The above process can be summarized as follows:

1. Calculate R_j for $j = 1,2, . . ,n$.
2. Calculate P_j for $j = 1,2, . . ,n$.
3. Calculate A_j for(j = 1,2, . . ,n.
4. Calculate Ratio (x_j) for $j = 1,2, . . ,n$.

5. Determine the variable x_k which maximizes Ratio. If Ratio (x_k) = 0, then go to step 7; otherwise, set $x_k = 1$.

6. Adjust the right hand side and return to Step 1.

7. Stop.

The second part of the procedure starts with the final feasible solution from the first part and improves on it by Method 5 of Procedure 1.

Chapter 3

Computational Experience

In order to evaluate and compare the three procedures described in Chapter 2, Pascal programs were written for each and run on a DEC20 system at Stanford University. The procedures were tested on 73 problems. Fifty seven of these were generated randomly, where 8 of these were of Type I, 16 were of Type II, 21 were of Type II' and 11 were of Type III. The types are as described in Table I, where the parameters are integers randomly generated for the indicated intervals.

Table I

DESCRIPTION OF THE RANDOMLY GENERATED TEST PROBLEMS

Parameter	Problem Type			
	I	II	II'	III
c_j'	[-20,80]	[0,100]	[0,100]	[0,100]
a_{ij}'	[-40,60]	[0,100]	[0,100]	[0,1]
b_i'	[50,200]	[400,1600]	[300,1200]	1
x_j	0-1	0-1	0-1	0-1

Letting m be the number of functional constraints and n the number of variables, eight problems of each type have m x n = 15x15, and the other are larger (such as 15x30, 30x15, 30x30, 60x30, 60x60, 60x120, 60x300). For the problems with n > 300, the range of the right hand side was changed to [4000,8000]. Seventeen of the problems tested were standard test problems in the literature--Haldi's IBM problems (#4 and #6) and nine Allocation Problems reproduced by

Trauth and Woolsey [32], four problems given by Petersen [23], two problems given by Senju and Toyoda [26], and four problems from Hillier [17]. These problems are denoted in the tables by Haldi, A, Pet, ST, and H respectively.

Table II presents a comparison of two definitions of Δ_i, (i) and (iii), and two Phase 1 methods. The last column of Table II shows the difference in the quality of the final solution obtained for each of these eight problems with each method in Phase 1 and each of the definitions of Δ_i. The measure of quality used throughout this chapter is the "normalized deviation" from the optimal solution $x^{(opt)}$, where the normalized deviation from optimality for a solution x is defined as

$$\frac{cx^{(opt)} - cx}{\sqrt{\sum_{j=1}^{n} c_j^2}},$$

where $x^{(opt)}$ has been obtained by Lindo. The geometrical interpretation of this quantity is that it is the Euclidean distance from x to the hyperplane $cx = cx^{(opt)}$.

The times given throughout this chapter are CPU times in seconds.

In Phase 1 of Procedure 1, Lindo has been used on the DEC20 System to obtain $x^{(1)}$ and $x^{(2)}$, as well as the basic feasible solutions generated in moving from $x^{(1)}$ to $x^{(2)}$. The times given under Lindo in each table are the times used by Lindo to obtain $x^{(1)}$ and $x^{(2)}$. Two definitions of Δ_i, (i) and (iii), have been

Table II

COMPARISON OF TWO DEFINITIONS OF Δ_I, (i) AND (iii)

AND TWO PHASE 1 METHODS

m	n	Problem type & number	Δ_1	Method	CPU time	Normalized Dev.
15	15	I-1	(i)	2a	6.42	0.109
15	15	I-1	(i)	2b	1.87	0.109
15	15	I-1	(iii)	2a	7.17	0.109
15	15	I-1	(iii)	2b	1.73	0.109
15	15	I-2	(i)	2a	2.99	0
15	15	I-2	(i)	2b	1.95	0
15	15	I-2	(iii)	2a	7.38	0
15	15	I-2	(iii)	2b	2.13	0
15	15	I-3	(i)	2a	12.9	0.248
15	15	I-3	(i)	2b	6.14	0.248
15	15	I-3	(iii)	2a	14.74	0.248
15	15	I-3	(iii)	2b	6.52	0.248
15	15	I-4	(i)	2a	13.38	0.108
15	15	I-4	(i)	2b	4.35	0.108
15	15	I-4	(iii)	2a	12.49	0.108
15	15	I-4	(iii)	2b	3.12	0.108
15	15	I-5	(i)	2a	15.49	0
15	15	I-5	(i)	2b	2.83	0
15	15	I-5	(iii)	2a	14.79	0
15	15	I-5	(iii)	2b	2.67	0
15	15	I-6	(i)	2a	15.70	0.363
15	15	I-6	(i)	2b	2.08	0.363
15	15	I-6	(iii)	2a	15.93	0.363
15	15	I-6	(iii)	2b	1.99	0.363
15	15	I-7	(i)	2a	16.80	0.063
15	15	I-7	(i)	2b	1.96	0.063
15	15	I-7	(iii)	2a	15.77	0.063
15	15	I-7	(iii)	2b	2.34	0.063
15	15	I-8	(i)	2a	12.52	0.284
15	15	I-8	(i)	2b	2.51	0.284
15	15	I-8	(iii)	2a	12.48	0.284
15	15	I-8	(iii)	2b	3.38	0.284
Average						0.147

tested on eight Type I problems. Even though the optimal value of r and the corresponding values of $x^{(2)}$ were different for each definition of Δ_i, the eventual solutions obtained by Procedure 1 were exactly the same for each problem. Therefore, only one definition of Δ_i was used in the rest of the testing process. The one chosen was the first definition (i), since it requires less computational effort. On the problems tested, Method 2b has been much faster and has given the same final solution from Procedure 1 as Method 2a, as suggested by Table II, so only Method 2b was used on the subsequent problems.

However, in general, Methods 2a and 2b do not necessarily lead to the same final solution. Furthermore, on problems where it is difficult to find a feasible solution in Phase 2, the chances of being successful should be better with Method 2a than 2b. Therefore, one can use Method 2a where a feasible solution is not found by method 2b. One explanation for the two methods giving the same final solution on the first eight test problems might be that in the 0-1 case, the basic feasible solutions obtained in getting $x^{(2)}$ might not be very different from $x^{(2)}$ when rounded. Therefore, rounded solutions used as the starting points for the Phase 2 searches for a feasible solution might not be very different for the two Phase 1 methods. One should also add that, in the general integer programming case, the situation would be different.

The three procedures have been compared according to the quality of their final solutions and their running times. A summary of the performance of these procedures is given in Tables III, IV, V and VI. Procedures 1,2, and 3 were run on 16 Type II problems and Table

III shows the resulting average normalized deviation from optimality and execution time for each problem, as well as the overall averages and the percentage of the problems for which an optimal solution is found. Even though Procedure 2 seems to be faster, Table III strongly suggests that its solutions tend to be inferior to those from Procedures 1 and 3 for this type of problem. Procedure 1 obtained the optimal solution 25% of the time, as compared to 31.3% for Procedure 3. Even though these are not very high percentages, the average normalized deviation from optimality in both cases was very low, 0.07 for Procedure 1 and 0.06 for Procedure 3. This suggests that solutions obtained by these procedures are, in general, very close to optimal. When the best solution for all three procedures were taken, the resulting solution was optimal 50% of the time. Therefore, another way of finding an approximate solution to a problem would be to run all three procedures on the problem and take the best solution obtained.

Another situation to be tested is the case where the problems have smaller feasible regions. Type II' problems are a modified form of Type II problems, where the range of the right hand side has been scaled down.

The three procedures were run on 18 Type II' problems, and the results are shown in Table IV. In comparison to Table III, the percentage of solutions that are optimal has actually increased from 25% to 27.8% in the case of Procedure 1. For this procedure, there is a very small increase in the average normalized deviation from optimality, from 0.07 to 0.08. The average normalized deviation from optimality for Procedure 3 is close to this, 0.09, but the percentage

Table III

SUMMARY OF PERFORMANCE FOR THE THREE PROCEDURES ON TYPE II PROBLEMS

m	n	Problem Type & no.	Procedure 1				Procedure 2			Procedure 3			Best Solution		
			Time		Norm Dev.	Optimal	Time	Norm dev.	Optimal	Time	Norm Dev.	Optimal	Optimal	Norm dev.	Proce-dure
			Lindo	Rest											
15	15	II-1	1.87	0.28	0.06	NO	1.04	0.39	NO	1.23	0.07	NO	NO	0.06	1
15	15	II-2	1.75	2.10	0	YES	1.48	0.290	NO	1.53	0	YES	YES	0	1,3
15	15	II-3	1.76	2.37	0.03	NO	2.24	0.60	NO	2.32	0.01	NO	NO	0.01	3
15	15	II-4	1.83	3.27	0	YES	3.11	0.25	NO	3.19	0.13	NO	YES	0	1
15	15	II-5	1.83	0.32	0.01	NO	0.17	0.46	NO	0.25	0.02	NO	NO	0.01	1
15	15	II-6	1.79	1.04	0.09	NO	0.49	0.27	NO	0.57	0	YES	YES	0	3
15	15	II-7	2.18	1.34	0.39	NO	1.19	0.05	NO	1.27	0	YES	YES	0	3
15	15	II-8	1.93	2.06	0.06	NO	1.51	0.23	NO	1.59	0	YES	YES	0	3
15	15	II-9	1.63	2.21	0.04	NO	1.23	0.20	NO	1.58	0.02	NO	NO	0.02	3
15	15	II-10	1.86	2.56	0.03	NO	2.36	0.42	NO	2.42	0.21	NO	NO	0.03	1
15	15	II-11	2.02	3.25	0.02	NO	3.13	0.03	NO	3.20	0.02	NO	NO	0.02	1,3
15	15	II-12	1.68	3.54	0	YES	3.40	0	YES	3.47	0.31	NO	YES	0	1,2
15	15	II-13	1.80	4.26	0	YES	3.13	0.07	NO	4.19	0.07	NO	YES	0	1
15	15	II-14	1.79	5.02	0.10	NO	4.48	0.22	NO	4.55	0.06	NO	NO	0.06	3
15	15	II-15	1.71	5.30	0.07	NO	5.18	0	YES	5.25	0	YES	YES	0	2,3
15	15	II-16	1.86	6.09	0.08	NO	5.47	0.07	NO	5.54	0.07	NO	NO	0.07	2,3
Average			1.84	2.82	0.07	25%	2.51	0.21	12.5%	2.63	0.06	31.3%	50%	0.01	

Table IV

SUMMARY OF PERFORMANCE FOR THE THREE PROCEDURES ON TYPE II' PROBLEMS

m	n	Problem Type & no.	Procedure 1				Procedure 2			Procedure 3			Best Solution		
			Time		Norm Dev.	Optimal	Time	Norm dev.	Optimal	Time	Norm Dev.	Optimal	Optimal	Norm dev.	Proce-dure
			Lindo	Rest											
15	15	II'-1	1.76	0.30	0	YES	0.19	0.46	NO	0.255	0	YES	YES	0	1,3
15	15	II'-2	1.79	0.28	0	YES	0.17	0.24	NO	0.242	0.07	NO	YES	0	1
15	15	II'-2	1.97	0.58	0.19	NO	0.45	0.27	NO	0.52	0.14	NO	NO	0.14	3
15	15	II'-4	1.86	1.27	0.19	NO	1.14	0	YES	1.21	0.18	NO	YES	0	2
15	15	II'-5	1.93	1.58	0	YES	1.45	0.37	NO	1.52	0.10	NO	YES	0	1
15	15	II'-6	2.15	2.29	0.40	NO	2.16	0.14	NO	2.23	0.02	NO	NO	0.02	3
15	15	II'-7	2.33	3.01	0.04	NO	2.48	0.36	NO	2.55	0.15	NO	NO	0.04	1
15	15	II'-8	2.02	3.32	0	YES	3.19	0.22	NO	3.25	0	YES	YES	0	1,3
15	15	II'-9	2.35	0.46	0.04	NO	0.24	0.33	NO	0.36	0.04	NO	NO	0.04	1,3
15	15	II'-10	2.52	1.32	0.04	NO	1.12	0.08	NO	1.23	0.12	NO	NO	0.04	1
30	15	II'-11	3.09	2.07	0	YES	1.55	0.30	NO	2.03	0.16	NO	YES	0	1
30	15	II'-12	2.72	2.46	0.18	NO	2.31	0.81	NO	2.39	0.07	NO	NO	0.07	3
30	30	II'-13	5.73	3.59	0.05	NO	3.33	0.62	NO	3.47	0.02	NO	NO	0.02	3
30	30	II'-14	5.03	5.08	0.04	NO	4.51	0.13	NO	5.05	0.11	NO	NO	0.04	1
30	60	II'-15	10.89	3.59	0.10	NO	3.07	0.49	NO	3.38	0.18	NO	NO	0.10	1
60	30	II'-16	22.61	6.51	0.13	NO	6.12	0.60	NO	3.38	0.18	NO	NO	0.13	1,3
60	60	II'-17	31.81	15.22	0.02	NO	13.54	0.39	NO	14.53	0.02	NO	NO	0.02	1,2,3
60	120	II'-18	107.00	7.16	0.10	NO	3.45	0.40	NO	6.03	0.13	NO	NO	0.10	1
60	300		272.40	786.7			475.05			650.00					
60	400		100.09	3681.1			1080.39			2760.09					
30	500		120.14	1012.52			1080.39			2160.01					
Average					0.08	27.8%		0.35	5.6%		0.09	11.1%	33.3%	0.04	

of optimal solutions has dropped from 31.3% to 11% for the Type II' problems. Procedure 2 has again done worse than these two. The results suggest that Procedure 3, in general, finds good approximate solutions for the problems, but Procedure 1 is more consistent in finding optimal solutions. It also suggests that Procedure 1 is not affected by the size of the feasible region for a problem. However, considerably more testing would be needed to draw statistically significant conclusions.

Procedure 3 cannot be used on Type I problems, so only Type II, II' and III problems can be used for comparing all three procedures. The results for Type III problems are given in Table V. The H series from Hillier [17] also are Type III problems. Contrary to the previous test results, Procedure 2 seems to do very well for this type of problem since it found the optimal solutions for 66.7% of the Type III problems. The other two procedures did quite well for this type of problems as well, namely, 40% and 26.7% for Procedures 1 and 3, respectively. The average normalized deviation from optimality was very small for all three procedures.

The reason for Procedure 2 doing so well for Type III problems and so poorly on Type II problems apparently is that Procedure 2 tries to assign the value of 1 to as many variables as possible. This strategy does not allow for further changes in the other variables. In Type III problems, only very few of the variables equal 1 in an optimal solution, so the Procedure 2 strategy works very well.

Comparing Tables II, III, IV and V, it can be deduced that all three procedures give better quality results on Type III problems.

Table V

SUMMARY OF PERFORMANCE FOR THE THREE PROCEDURES ON TYPE III PROBLEMS

m	n	Problem Type & no.	Procedure 1				Procedure 2			Procedure 3			Best Solution		
			Time		Norm Dev.	Optimal	Time	Norm dev.	Optimal	Time	Norm Dev.	Optimal	Optimal	Norm dev.	Procedure
			Lindo	Rest											
15	15	III-1	1.56	8.07	0	YES	7.47	0	YES	7.56	0	YES	YES	0	1,2,3
15	15	III-2	1.58	8.51	0	YES	8.29	0	YES	8.39	0.10	NO	YES	0	1,2
15	15	III-3	1.59	9.25	0.03	NO	9.08	0	YES	9.17	00.3	NO	YES	0	2
15	15	III-4	1.64	10.01	0.04	YES	9.43	0	YES	9.53	0.04	NO	YES	0	2
15	15	III-5	1.70	11.58	0	YES	11.37	0	YES	11.47	0.04	NO	YES	0	1,2
15	15	III-6	1.69	12.41	0.04	NO	12.21	0.04	NO	12.30	0.07	NO	NO	0.04	1,2
15	15	III-7	1.58	13.38	0.01	NO	13.02	0	YES	13.09	0.01	NO	YES	0	2
15	15	III-8	1.58	14.24	0.09	NO	14.06	0.02	NO	14.16	0.08	NO	NO	0.02	2
15	30	III-9	2.03	15.56	0.06	NO	15.02	0.16	NO	15.20	0.11	NO	NO	0.06	1
30	15	III-10	2.07	16.40	0	YES	16.18	0	YES	16.28	0	YES	YES	0	1,2,3
30	30	III-11	3.75	18.02	0.06	NO	17.14	0.06	NO	17.33	0.08	NO	NO	0.06	1,2
15	15	H-2	1.75	2.05	0.01	NO	1.31	0	YES	1.47	0.01	NO	YES	0	2
15	15	H-3	1.65	0.32	0.06	NO	0.15	0.17	NO	0.24	0.06	NO	NO	0.06	1,3
15	15	H-4	0.53	1.08	0	YES	0.53	0	YES	1.00	0	YES	YES	0	1,2,3
15	15	H-5	1.64	1.34	0	YES	1.22	0	YES	1.30	0	YES	YES	0	1,2,3
Average					0.03	40%		0.03	66.7%		0.04	26.7%	66.7%	0.016	

Procedure 1 seems to be more consistent than the others in its quality of results for different types of problems.

The growth of execution time for each procedure on larger problems ($n > 15$) can be seen in Tables IV and V. Procedures 2 and 3 solved problems with $n \leq 120$ in less than 10 seconds for all but one problem. The execution times for Procedure 1 tend to be considerably larger, but it still was less than 2 minutes for a problem with $n = 120$. In general, for the three problems with $n \geq 300$ the execution time did not increase rapidly (if at all) as n was increased. Because of the size of these problems, optimal solutions were not obtained. Therefore, no normalized deviations are given for these problems.

Table VI shows the changes in the objective function value (Z) in different parts of the three procedures. Z_1 is the objective function value at the end of Phase 2 for Procedure 1. In Procedures 2 and 3, it is the objective function value at the end of the first part of these procedures, before applying Method 5 of Phase 3. Using the labeling of parts for Method 5 given in [16,18], Z_2, Z_4, Z_5, Z_6 and Z_7 are the objective function values at the end of parts 2,4,5,6 and 7, respectively, in the last iteration (if any) where an improvement was obtained in that part. Z_9 corresponds to the objective function value obtained at the end of the Phase 2 type of search in Method 5. Table VI shows that the solutions were very rarely improved in parts 4,5,6 and 7, whereas the Phase 2 type of search of Method 5 improved the results more than 25% of the time. More improvements were made on Z_1 in Procedures 1 and 3 than in Procedure 2. This strengthens the argument that once variables are

Table VI

CHANGES IN THE OBJECTIVE FUNCTION VALUE IN DIFFERENT PARTS OF THE PROCEDURES

Problem Type & number	Procedure 1							Norm dev. from opt.	Procedure 2							Norm dev. from opt.	Procedure 3							Norm dev. from opt.
	Z_1	Z_2	Z_4	Z_5	Z_6	Z_7	Z_9		Z_1	Z_2	Z_4	Z_5	Z_6	Z_7	Z_9		Z_1	Z_2	Z_4	Z_5	Z_6	Z_7	Z_9	
II-9	499	—	—	—	—	—	→	0.044	471	—	—	—	—	—	→	0.198	506	—	—	—	—	—	→	0.024
II-10	498	—	—	—	—	—	→	0.025	420	—	—	—	—	—	→	0.419	462	—	—	—	—	—	→	0.207
II-11	465	—	—	—	—	→	471	0.016	469	—	—	—	—	—	→	0.026	471	—	—	—	—	—	→	0.016
II-12	667	—	—	—	—	—	→	0	667	—	—	—	—	—	→	0	592	—	—	—	—	—	→	0.305
II-13	551	—	—	—	—	—	→	0	537	—	—	—	—	—	→	0.073	537	—	—	—	—	—	→	0.073
II-14	475	484	—	—	—	—	→	0.107	463	—	—	—	—	—	→	0.220	493	—	—	—	—	—	→	0.059
II-15	653	—	—	—	—	→	664	0.067	678	—	—	—	—	—	→	0	678	—	—	—	—	—	→	0
II-16	600	—	—	—	—	—	→	0.077	602	—	—	—	—	—	→	0.067	602	—	—	—	—	—	→	0.067
II'-1	381	—	—	—	—	→	446	0	365	—	—	—	—	—	→	0.463	446	—	—	—	—	—	→	0
II'-2	477	533	—	—	—	→	536	0	475	—	—	—	—	—	→	0.243	518	—	—	—	—	—	→	0.071
II'-3	506	512	—	—	—	—	→	0.185	494	—	—	—	—	—	→	0.271	522	—	—	—	—	—	→	0.137
II'-4	497	—	—	—	—	—	→	0.194	536	—	—	—	—	—	→	0	499	—	—	—	—	—	→	0.184
II'-5	552	—	—	—	—	—	→	0	467	—	—	—	—	—	→	0	528	—	—	—	—	—	→	0.103
II'-6	455	—	—	—	—	—	→	0.400	505	—	—	—	—	—	→	0.143	529	—	—	—	—	—	→	0.020
II'-7	451	476	—	—	—	—	→	0.043	403	—	—	—	—	—	→	0.359	452	—	—	—	—	—	→	0.147
II'-8	530	—	—	—	—	—	→	0	476	—	—	—	—	—	479	0.221	530	—	—	—	—	—	→	0
II'-9	616	621	—	—	—	—	→	0.039	533	—	—	—	—	—	→	0.331	621	—	—	—	—	—	→	0.039
II'-10	722	—	—	—	—	—	→	0.041	709	—	—	—	—	—	→	0.079	696	—	—	—	—	—	→	0.118

Table VI
(continued)

Problem Type & number	Procedure 1							Norm dev. from opt.	Procedure 2							Norm dev. from opt.	Procedure 3							Norm dev. from opt.
	Z_1	Z_2	Z_4	Z_5	Z_6	Z_7	Z_9		Z_1	Z_2	Z_4	Z_5	Z_6	Z_7	Z_9		Z_1	Z_2	Z_4	Z_5	Z_6	Z_7	Z_9	
II'-11	384	→	→	→	→	→	406	0	345	→	→	→	→	→	→	0.301	374	→	→	→	→	→	→	0.157
II'-12	392	→	→	→	→	→	→	0.177	278	→	→	→	→	→	→	0.810	412	→	→	→	→	→	→	0.066
II'-13	600	→	→	→	→	→	607	0.046	424	→	→	→	→	→	→	0.619	617	→	→	→	→	→	→	0.015
II'-14	455	→	→	→	→	→	474	0.040	448	→	→	→	→	→	→	0.128	454	→	→	→	→	→	→	0.108
II'-15	695	→	→	→	→	→	→	0.101	429	→	→	526	→	→	→	0.492	662	→	→	→	→	→	→	0.178
II'-16	509	→	→	→	→	→	→	0.126	362	→	→	→	→	→	→	0.601	542	→	→	→	→	→	→	0.019
II'-17	682	→	→	→	→	→	→	0.015	518	→	→	→	→	→	→	0.385	678	→	→	→	→	→	→	0.024
II'-18	770	781	→	→	→	→	→	0.101	597	→	→	→	→	→	→	0.397	719	→	→	→	→	→	762	0.132
III-1	96	→	→	→	→	→	→	0	96	→	→	→	→	→	→	0	96	→	→	→	→	→	→	0
III-2	92	→	→	→	→	→	→	0	92	→	→	→	→	→	→	0	70	→	→	→	→	→	→	0.103
III-3	88	→	→	→	→	→	→	0.0275	94	→	→	→	→	→	→	0	88	→	→	→	→	→	→	0.0275
III-4	87	→	→	→	→	→	89	0.035	98	→	→	→	→	→	→	0	87	→	→	→	→	→	89	0.035
III-5	93	→	→	→	→	→	→	0	93	→	→	→	→	→	→	0	84	→	→	→	→	→	→	0.036
III-6	98	→	→	→	→	→	→	0.36	98	→	→	→	→	→	→	0.036	91	→	→	→	→	→	→	0.067
III-7	75	→	→	→	→	→	82	0.010	84	→	→	→	→	→	→	0	75	→	→	→	→	→	82	0.010
III-8	77	→	→	→	→	→	→	0.094	98	→	→	→	→	→	→	0.018	77	→	→	→	→	→	→	0.094
III-9	48	123	→	→	→	→	→	0.060	94	→	→	→	→	→	→	0.163	100	→	→	→	→	→	108	0.113
III-10	0	58	→	→	→	→	→	0	58	→	→	→	→	→	→	0	58	→	→	→	→	→	→	0
III-11	0	95	→	→	→	→	→	0.056	95	→	→	→	→	→	→	0.056	88	→	→	→	→	→	→	0.079
Total no. of changes	9	0	0	0	0	7			0	0	1	0	0	1			0	0	0	0	0	4		

set to 1 in Procedure 2, they do not readily allow further improvements. Procedure 3 in its present form gives very good solutions and is very fast. Furthermore, if parts 2-7 of Method 5 are removed, the algorithm will become much faster. On average, this should not decrease the quality of the results significantly. For all three procedures, the quality of the final solutions perhaps can be improved by increasing the number of iterations allowed in Phase 3 or by trying the second part of Method 4 (without Method 1) in Phase 3.

Because all three procedures continue with the identical method (Method 5 of Phase 3) after obtaining Z_1, the Z_1 columns of Table VI provide a direct comparison of the parts that differ. This comparison again suggests that Procedure 2 is quite inferior to the others for Type II and II' problems, but probably the best for Type III problems, where Procedure 1 and 3 perform about the same for all the types.

Table VII gives test results on some standard problems from the literature. The A series problems are single constraint allocation problems. They were designed to test the sensitivity of algorithms to small changes in the right hand side of the problem. Therefore, the nine problems are the same except for their right hand sides. For two of these problems, A-5 and A-9, Lindo had found the optimal integer solution as $x^{(1)}$, so Procedure 1 was not tested on these. The best solution obtained by the three procedures was optimal in five out of the nine problems. Two Haldi problems were only tested on Procedure 1 because the right hand side and the A matrix have negative elements. Even though Procedure 1 found the closest

Table VII

SUMMARY OF PERFORMANCE FOR THE THREE PROCEDURES ON STANDARD TEST PROBLEMS

m	n	Problem Type & no.	Procedure 1				Procedure 2			Procedure 3			Best Solution		
			Time		Norm Dev.	Optimal	Time	Norm dev.	Optimal	Time	Norm Dev.	Optimal	Optimal	Norm dev.	Procedure
			Lindo	Rest											
10	20	Pet-4	1.85	0.35	0.03	NO	0.20	0.73	NO	0.27	0.732	NO	NO	0.03	1
10	28	Pet-5	2.22	1.14	0.008	NO	0.55	0.90	NO	1.07	0.50	NO	NO	0.008	1
5	39	Pet-6	2.21	2.16	0.46	NO	1.54	0.55	NO	2.07	0.55	NO	NO	0.46	1
5	30	Pet-7	2.48	3.40	0.01	NO	3.15	0.73	NO	3.29	0.36	NO	NO	0.01	1
15	15	Haldi-4			0.258	NO	--	--	--	--	--	--	NO	0.258	1
31	31	Haldi-6	6.40	1.01	0.179	NO	--	--	--	--	--	--	NO	0.179	1
1	10	A-1			0.200	NO		1.25	NO		0.200	NO	NO	0.125	2
1	10	A-2			0	YES		0	YES		0	YES	YES	0	1,2,3
1	10	A-3			0	YES		0	YES		0.125	NO	YES	0	1,2
1	10	A-4			0.25	NO		0.125	NO		0.25	NO	NO	0.125	2
1	10	A-5			--	--		0	YES		0	YES	YES	0	1,2
1	10	A-6			0	YES		0	YES		0	YES	YES	0	1,2,3
1	10	A-7			0.05	NO		0.05	NO		0.05	NO	NO	0.05	1,2,3
1	10	A-8			1.75	NO		0	NO		0.175	NO	NO	0.075	2
1	10	A-9			--	--		0	YES		0	YES	YES	0	2,3
30	60	ST A	15.26	15.29	0.587	NO	8.508	0.730	NO	9.256	0.073	NO	NO	0.073	3
30	60	ST B	12.08	6.409	0.07	NO	15.448	0.258	NO	16.168	0.0385	NO	NO	0.0385	3

possible approximate solution to the optimal solution, the normalized deviation from optimality is large because the objective row coefficients are small (all l's). In the Pet and ST series, even though the solutions obtained were not optimal, the best solutions obtained from all three procedures have small normalized deviations from optimality.

Table VIII give a comparison of the best solution obtained by all three procedures (fourth column) with the solution obtained by the pivot and complement algorithm developed by Balas and Martin [2].

Table VIII

COMPARISON WITH BALAS-MARTIN ALGORITHM

Problem	m	n	Best Solution	Balas-Martin
			$\frac{\lvert z_{opt} - z_{neu} \rvert}{\lvert z_{opt} \rvert}$	$\frac{\lvert z_{opt} - z_{neu} \rvert}{\lvert z_{opt} \rvert}$
PET-4	10	20	0.017	0
PET-5	10	28	0.003	0
PET-6	5	39	0.240	0.0028
PET-7	5	50	0.005	0.0023
ST A	30	60	0.021	0
ST B	30	60	0.010	0

Chapter 4

Conclusions

A heuristic algorithm aims at obtaining a very good feasible solution relatively quickily. Although the primary motivation of the present algorithms was to provide an efficient way of dealing with the frequently encountered integer programming problems that are beyond the computational capability of exact algorithms, heuristic algorithms also can be useful on smaller problems by providing an advanced starting solution to accelerate an exact algorithm.

This thesis presents three heuristic procedures for certain classes of Binary Integer Programming problems. The construction of the procedures was given in Chapter 2. These procedures can be used to efficiently obtain a very good (but not necessarily optimal) solution for problems that are too large to be solved exactly. In fact, test proablems with up to 500 variables have been successfully run with only modest exception times. For smaller problems, they can be used to obtain a good starting solution for the exact algorithm.

The procedures were tested on different types of problems to evaluate their effectiveness and efficiency, as reported in Chapter 3. The procedures have tended to perform differently for different types of problems. Procedure 2 tends to give better quality solutions for Type III problems, while quite consistently doing worse than the other two for Type II problems. Even though Procedures 1 and 3 seemed to have similar performances on most types of problems, Procedure 1 seemed to be slightly superior to Procedure 3 on the average regarding the quality of the final solution. However, Procedure 1 is somewhat slower than the other two. More testing needs to be done to obtain statistically significant comparisons.

Solving a problem by all three procedures and taking the best solution is a promising method. The execution time for all three of these procedures should be relatively insignificant, compared to the time needed by an exact algorithm for large problems.

Parts 4 to 7 of Phase 3 (used in all three procedures) very rarely improved the results. Therefore, these parts can be deleted from this phase, which will significantly decrease execution time.

In Phase 1 of Procedure 1, it appears that the first definition of Δ_i and Method 2b are appropriate choices.

Method R1-R3-5 of [18] had given very powerful results. This is another combination of methods that can be tried for the 0-1 integer programming case. Only six test problems were available for comparing Balas and Martin's pivot and complement algorithm with these three procedures, but the limited results strongly suggest that the pivot and complement algorithm is superior in the quality of the solutions obtained. More testing needs to be done for a definitive comparison of effectiveness on different types of problems. No comparison of the execution times was made since testing was done on different computers and in different programming languages.

One important area for future research would be to extend these heuristic algorithms to mixed integer programming.

APPENDIX

This appendix presents the Pascal code for Procedure 1. The labeling of different parts and phases are in accordance with [16,18].

```
 *****************************************************************)
 *                                                               *)
.*  The purpose of this program is to find a good approximate    *)
(*solution to the following Binary Integer Programming problem:  *)
(*          Max  Z = Cx,                                         *)
(*          subject to :                                         *)
(*                     Ax <= B                                   *)
(*                      x = 0 or 1                               *)
(*                                                               *)
(*          where: The dimension of A is m by n                  *)
(*                 The dimension of B is m                       *)
(*                 The dimension of C is n                       *)
(*          and the set of feasible solutions is assumed to have *)
(*          an interior point.                                   *)
(*****************************************************************)


(*$X+*)
PROGRAM HEURIS(INFILE,OUTFILE);

TYPE ROWS=ARRAY[1..500]OF REAL;
      COLS=ARRAY[1..500]OF REAL;
      MATRIX=ARRAY[1..500,1..500]OF INTEGER;
       PLMATRIX=ARRAY[1..500,1..500]OF REAL;
      RLRATIO=ARRAY[1..500]OF REAL;
      INTCOLS=ARRAY[1..500]OF INTEGER;
       INTROWS=ARRAY[1..500]OF INTEGER;

VAR INFILE,OUTFILE:TEXT;
     NEWC,XBF,MATRIXA,D:PLMATRIX;
     TEMPC,OBJROW :INTCOLS;
     U,UPRIME :INTROWS;
     IROWS,SPRIME,S,ORGRHS,RHS:ROWS;
     X1,X2,TEMPX,QSTAR,AP    :COLS;
     R : RLRATIO;
     DELX,LETZ,ORDER,NEWO,RANGE,ELIGIBLE:INTCOLS;

     L,LPRIME,O:INTROWS;
        NEWR,PPRIME:MATRIX;
X,XF,XL,DELTA:INTCOLS;
     SUM,COUNTER,A,TOTLINES,NUM,P,H,G,MIN,Z,ZVAL,F,E,NO,M,N,B,C,
             COUNT1,COUNT,I,J,K,INDEX,T,OBJVAL:INTEGER;
     CONT,POSSIBLE,FOUND,STP10,ENDPHASE2,TERMINATE,SAME,IMPROVED,
         CHECKJ,STP5,STP6,STP7,INVESTIGATE,ENDPART2,ENDPART3,ENDPART4,
         ENDPART5,ENDPART6,ENDPART7,INFEAS:BOOLEAN;
     SITERM,TERM,DEV,ALFA,SUM2,ZLIN:REAL;
     CRITERION:CHAR;


(*  Read the problem                                             *)



PROCEDURE READPROB(VAR CRITERION:CHAR;VAR MATRIXA:PLMATRIX;
     VAR OBJROW:INTCOLS;VAR M,N:INTEGER;VAR RHS,ORGRHS:ROWS);

VAR I,J,COL,ROW:INTEGER;

BEGIN
     READLN(INFILE,CRITERION);
     READLN(INFILE,N);
     READLN(INFILE,M);
     FOR J:=1 TO N DO
       READ(INFILE,OBJROW[J]);
     READLN(INFILE);
     FOR ROW:=1 TO M DO
       BEGIN
           FOR COL:=1 TO N DO
             BEGIN
                  READ(INFILE,MATRIXA[ROW,COL]);
             END;
           READLN(INFILE);
        END;
      FOR I:=1 TO M DO
        BEGIN
            READ(INFILE,RHS[I]);
            ORGRHS[I]:=RHS[I];
        END;
       READLN(INFILE);
     READLN(INFILE);
END;
(*  Read the solution of LP relaxation                                      *)

PROCEDURE READX1(VAR ZLIN:REAL;VAR X1:COLS);
VAR J:INTEGER;
BEGIN
     READLN(INFILE,ZLIN);
     FOR J:=1 TO N DO
       READ(INFILE,X1[J]);
     READLN(INFILE);
     READLN(INFILE);
END;


(*  Read Basic Feasible Solutions in getting X2, starting from X1 *)
(*  Totlines is the number of basic feasible solutions read. XBF  *)
(*  is the matrix formed by all the basic feasible solutions together.
PROCEDURE READBFSOLN(VAR XBF:PLMATRIX;VAR TOTLINES:INTEGER);
VAR I,J:INTEGER;
BEGIN
     I:=0;
     WHILE NOT(EOF(INFILE)) DO
       BEGIN   I:=I+1;
          FOR J:=1 TO N DO

             READ(INFILE,XBF[I,J]);
          READLN(INFILE);
       END;
     TOTLINES:=I;
END;


(*  Make the necessary adjustments according to how many basic     *)
(*  feasible solutions read.                                        *)

PROCEDURE ADJUST(TOTLINES:INTEGER;X1:COLS;XBF:PLMATRIX;NUM:INTEGER;
      VAR X2:COLS);

VAR J:INTEGER;
   TEMP1,TEMP2:INTCOLS;
```

```
    OK:BOOLEAN;
BEGIN
    OK:=FALSE;
    WHILE NOT(OK) AND (NUM<=TOTLINES) DO
       BEGIN
          FOR J:=1 TO N DO
            BEGIN
               TEMP1[J]:=TRUNC(X1[J]);
               X2[J]:=XPP[NUM,J];
               TEMP2[J]:=TRUNC(X2[J]);
               IF NOT(TEMP1[J] = TEMP2[J]) THEN
                    OK:=TRUE;
            END;
            IF NOT(OK) THEN
               NUM:=NUM+1;
       END;
END;
```

```
(*  Step 1 of Phase 2                                                    *)

PROCEDURE STEP1(VAR ALFA:REAL);
(*  Initialize Alpha                                                     *)
BEGIN
    ALFA:=0;
END;


(*  Step 2 of Phase 2                                                    *)

PROCEDURE STEP2(VAR TEMPX:COLS;X1,X2:COLS;ALFA:REAL);
(*  Set x = ( 1 - Alpha ) x1 + Alpha x2                                  *)
VAR J:INTEGER;
BEGIN
    FOR J:=1 TO N DO
      TEMPX[J]:=(1-ALFA)*X1[J] + ALFA*X2[J];
END;


(*  Step 3 of Phase 2                                                    *)

PROCEDURE STEP3( TEMPX:COLS;VAR Y:INTCOLS);
(*  Takes the scientific rounding of x                                   *)
VAR J:INTEGER;
BEGIN
    FOR J:=1 TO N DO
      BEGIN
         X[J]:=TRUNC(TEMPX[J] + 0.5);
         IF (X[J] < 0) THEN
              X[J]:=0;
      END;
END;
```

```
(*  Step 4 of Phase 2                                                 *)

PROCEDURE STEP4(VAR SUMQ:REAL;VAR QROWS:ROWS;MATRIXA:RLMATRIX;
    X:INTCOLS);
(*  Find the slack for each inequality and compute the infeasibility *)
VAR I:INTEGER;
BEGIN
    SUMQ:=0;
    FOR I:=1 TO M DO
      BEGIN
          QROWS[I]:=0;
         FOR J:=1 TO N DO
            QROWS[I]:=QROWS[I] + (MATRIXA[I,J] * X[J]);
         QROWS[I]:=QROWS[I] - RHS[I];
         IF (QROWS[I] > 0) THEN
            SUMQ:=SUMQ + QROWS[I];
      END;
END;


(*  Step 6 of Phase 2                                                *)

PROCEDURE STEP6(X:INTCOLS;VAR MATRIXA, NEWQ:RLMATRIX; VAR QSTAR:COLS;
      VAR DELX:INTCOLS;QROWS:ROWS);
VAR I,J:INTEGER;
      S:COLS;
BEGIN
    FOR J:=1 TO N DO
      BEGIN
(*  Compute Sj for each j                                            *)
         S[J]:=0;
         QSTAR[J]:=0;
         FOR I:=1 TO M DO
           IF (QROWS[I]>0) THEN
              S[J]:=S[J] + MATRIXA[I,J];
(*  Compute how much each xj shoul be increased in the favorable *)
(*  direction.                                                       *)
         IF (S[J]<0) AND (X[J]<1) THEN
            DELX[J]:=1
         ELSE
            IF ( S[J] > 0 ) AND ( X[J] > 0) THEN
               DELX[J]:=-1
            ELSE
               DELX[J]:=0;
(*  Compute qij and qj                                               *)
          FOR I:=1 TO M DO
         NEWQ[I,J]:=QROWS[I] + (MATRIXA[I,J] * DELX[J]);
         FOR I:=1 TO M DO
           IF (NEWQ[I,J] > 0 ) THEN
              QSTAR[J]:=NEWQ[I,J] + QSTAR[J];
      END;
END;


(*  Step 5 of Phase 2                                                *)

PROCEDURE STEP5(VAR NEWQ:RLMATRIX;QROWS:ROWS;VAR DELX,XF:INTCOLS;
       X:INTCOLS;VAR MATRIXA:RLMATRIX;VAR QSTAR:COLS;SUMQ:REAL;
       VAR FOUND,ENDPHASE2:BOOLEAN);
```

```
(*  Check if the round solution is feasible. If it is feasible,   *)
(*  this becomes the current feasible solution, otherwise go to step 6.
    VAR  J:INTEGER;
BEGIN
     IF (SUMI > 0) THEN
        STEP6(X,MATRIXA,NEWI,QSTAR,DELX,QROWS)
     ELSE
         BEGIN
              FOUND:=TRUE;
            ENDPHASE2:=TRUE;
            FOR J:=1 TO N DO
                   XF[J]:=X[J];
          END;
END;


(*  Step 10 of Phase 2                                              *)

PROCEDURE STEP10(X1,X2:COLS;VAR ALFA:REAL;VAR STP10:BOOLEAN);
(*  Reset the value of Alpha. Alpha gets the smallest value that *)
(*  will give the next different rounded x.                      *)
VAR J:INTEGER;
     MIN:REAL;
      THETA:COLS;
BEGIN
     MIN:=10000000;
     FOR J:=1 TO N DO
       BEGIN
            IF (X1[J]>=0.5) THEN
             BEGIN
              IF (X2[J]-X1[J] < 0) THEN   BEGIN
                 THETA[J]:=-((X1[J] - 0.5) / (X2[J] - X1[J]))+0.0001
             END
           ELSE
                  THETA[J]:=1.5;
             END
           ELSE
             BEGIN
              IF (X2[J] -X1[J] > 0) THEN   BEGIN
                 THETA[J]:=((0.5 - X1[J]) / (X2[J]- X1[J]))+0.0001
              END
           ELSE
                THETA[J]:=1.5;
               END;
            IF (THETA[J] < MIN) THEN
                 MIN:=THETA[J];
       END;
            ALFA:=MIN;
       STP10:=FALSE;
END;


(*  Step 7 of Phase 2                                               *)

PROCEDURE STEP71(VAR STP10:BOOLEAN;VAR ALFA:REAL;VAR QSTAR:COLS;
        VAR SUMI:REAL;VAR LETO:INTCOLS);
(*  Find the variables that can improve the infeasibility         *)
VAR P,L,COUNT,J:INTEGER;
BEGIN
     L:=0;
     FOR J:=1 TO N DO
      BEGIN
           IF ( QSTAR[J] < SUMQ ) THEN
              BEGIN
                  L:=L+1;
                  LETQ[L]:=J;
              END;
      END;
     IF (LETQ[1] = 0) THEN
         STP10:=TRUE;
END;


(*  The step replacing step 7 of Phase 2, when this phase is used *)
(*  in Method 5 of Phase 3                                        *)

PROCEDURE STEP7(OBJROW:INTCOLS;CRITERION:CHAR;LETQ:INTCOLS;
       VAR K:INTEGER;VAR SUMQ:REAL;QSTAR:COLS;DELX:INTCOLS);
(*  Find the variable which will make the largest improvement     *)
VAR I,T,J:INTEGER;
  MAXP:REAL;
    P:COLS;
BEGIN
    MAXP:=-10000000;
    FOR T:=1 TO N DO
      BEGIN
           IF NOT(LETQ[T]=0) THEN
             BEGIN
              J:=LETQ[T];
           IF (CRITERION = 'A' ) THEN
             P[J]:=SUMQ -QSTAR[J]
           ELSE
             P[J]:=(OBJROW[J]*DELX[J]) / (SUMQ - QSTAR[J]);
          IF ( P[J] > MAXP ) THEN
            BEGIN
                MAXP:=P[J];
                    K:=J;
            END;
             END;
      END;
END;


(*  Find the variable which will make the largest improvement without
(*  using Q.                                                       *)

PROCEDURE STEP7C(OBJROW:INTCOLS;CRITERION:CHAR;VAR K:INTEGER;
    VAR SUMQ:REAL;QSTAR:COLS;DELX:INTCOLS);
VAR J:INTEGER;
  MAXP:REAL;
    P:COLS;
BEGIN
    MAXP:=-1000;
    FOR J:=1 TO N DO
      BEGIN
           IF (CRITERION = 'A' ) THEN
             P[J]:=SUMQ -QSTAR[J]
           ELSE
             P[J]:=(OBJROW[J]*DELX[J]) / (SUMQ - QSTAR[J]);
          IF ( P[J] > MAXP ) THEN
```

```
                BEGIN
                    IF NOT(P[J]=0) THEN
                      BEGIN
                    MAXP:=P[J];
                       K:=J;
                      END
                END;
          END;
END;


(*  Step 9 of Phase 2                                                  *)

PROCEDURE STEP9(VAR K:INTEGER;VAR X,DELX:INTCOLS;VAR QROWS:ROWS;
       NEWQ:PLMATRIX;QSTAR:COLS;VAR SUMQ:REAL);
(*  Reset the value of x, qi and q                                     *)
VAR I:INTEGER;
BEGIN
    X[K]:=X[K]-DELX[K];
    FOR I:=1 TO M DO
       QROWS[I]:=NEWQ[I,K];
    SUMQ:=QSTAR[K];
END;


(*  Step 8 of Phase 2                                                  *)

PROCEDURE STEP8(OBJROW:INTCOLS;CRITERION:CHAR;QSTAR:COLS;
     NEWQ:PLMATRIX;VAR DELX, X:INTCOLS;VAR QROWS:ROWS;VAR SUMQ:REAL;
     LETQ:INTCOLS;VAR K:INTEGER);
BEGIN
    IF (LETQ[2] = 0) THEN
         BEGIN
         K:=LETQ[1];
             STEP9(K,X,DELX,QROWS,NEWQ,QSTAR,SUMQ);
          END
    ELSE
        BEGIN
        STEP7(OBJROW,CRITERION,LETQ,K,SUMQ,QSTAR,DELX);
         STEP9(K,X,DELX,QROWS,NEWQ,QSTAR,SUMQ);
        END;
END;


(*  Step 11 of Phase 2                                                 *)

PROCEDURE STEP11(ALFA:REAL;VAR TERMINATE:BOOLEAN);
(*  Check if Alpha <= 1                                                *)
BEGIN
    IF(ALFA <= 1) THEN
        TERMINATE:=FALSE
    ELSE
        TERMINATE:=TRUE;
END;



(*  Swap the values of two integer variables                           *)
PROCEDURE SWAP(VAR FIRST,SECOND:INTEGER);
VAR TEMP:INTEGER;
```

```
BEGIN
    TEMP:=FIRST;
    FIRST:=SECOND;
    SECOND:=TEMP;
END;


(*  Print the results in a file called Outfile                        *)

PROCEDURE RESULTS(VAR X,ELIGIBLE:INTCOLS;OBJROW:INTCOLS;Z ,
     PROW:INTEGER);

VAR I,J:INTEGER;
BEGIN
    FOR J:=1 TO N DO
       WRITE(OUTFILE,X[J]:3,' ');
    WRITELN(OUTFILE);
    WRITELN(OUTFILE);
    Z:=0;
    FOR J:=1 TO N DO
          Z:=Z+(X[J]*OBJROW[J]);
    WRITELN(OUTFILE,Z);
    WRITELN(OUTFILE,PROW);
END;


(*  Sort the array of indexes of variables, according to their *)
(*  objective row coefficients, from largest to smallest        *)

PROCEDURE SORT(VAR OBJROW,ELIGIBLE:INTCOLS;COUNT1:INTEGER);
VAR LARGEST,A,B,P,Q:INTEGER;
BEGIN
    FOR P:=1 TO (COUNT1-1) DO
      BEGIN
          LARGEST:=P;
          FOR Q:=(P+1) TO COUNT1 DO
           BEGIN
              A:=ELIGIBLE[Q];
              B:=ELIGIBLE[LARGEST];
              IF (OBJROW[A]>OBJROW[B]) THEN
                     LARGEST:=Q;
            END;
              SWAP(ELIGIBLE[P],ELIGIBLE[LARGEST]);
            END; END;


(*  Step 1 of Part 1 in Phase I                                   *)

PROCEDURE PART11(VAR X,XF:INTCOLS);
(*  Set x = xf                                                    *)
VAR J:INTEGER;
BEGIN
    FOR J:=1 TO N DO
        X[J]:=XF[J];
END;

(*  Sort elements of an array of integers from largest to smallest *)

PROCEDURE SORTIT(VAR LIST:INTCOLS;LENGTH:INTEGER);
VAR COUNT1,COUNT2,SMALLEST:INTEGER;
```

```
BEGIN
    FOR COUNT1:=1 TO (LENGTH-1) DO
      BEGIN
          SMALLEST:=COUNT1;
          FOR COUNT2:=(COUNT1+1) TO LENGTH DO
             IF LIST[COUNT2] < LIST[SMALLEST] THEN
                 SMALLEST:=COUNT2;
             SWAP (LIST[COUNT1],LIST[SMALLEST]);
      END
END;


(*  Step 2 of Part 1 of Phase 3                                       *)

PROCEDURE PART12(VAR ORDER, OBJROW,TEMPC,DELTA:INTCOLS;VAR NO:INTEGER)
VAR L,J:INTEGER;
BEGIN
(*  Order the non-zero objective row coefficients
    FOR J:=1 TO N DO
       IF (OBJROW[J] < 0) THEN
          TEMPC[J]:=-1*OBJROW[J]
       ELSE
          TEMPC[J]:=OBJROW[J];
    SORT(TEMPC,ORDER,N);
    NO:=0;
(*  The favorable change for variables with cj > 0, is set to 1 *)
(*  and variables with cj < 0 is set to -1                        *)
    FOR J:=1 TO N DO
       IF (TEMPC[J] > 0 ) THEN
          NO:=NO+1;
    FOR L:=1 TO NO DO
       BEGIN
         J:=ORDER[L];
       IF (OBJROW[J] > 0 ) THEN
             DELTA[J]:=1
       ELSE
          IF (OBJROW [J] < 0) THEN
               DELTA[J]:=-1;
       END;
END;


(* Step 3 of Part 1 of Phase 3                                        *)

PROCEDURE PART13(VAR S:ROWS;RHS:ROWS;MATRIXA:RLMATRIX;X:INTCOLS);
(* Find the slack for each inequality                                 *)
VAR I,J:INTEGER;
    SUM:REAL;
BEGIN
    FOR I:=1 TO M DO
      BEGIN
          SUM:=0;
          FOR J:=1 TO N DO
            SUM:=(MATRIXA[I,J]*X[J])+SUM;
          S[I]:=RHS[I]-SUM;
      END;
END;


(*  Step 1 of Part 2 of Phase 3                                       *)
```

```
PROCEDURE PART21(NO:INTEGER;S:ROWS;ORDER,OBJROW,X:INTCOLS;
    MATRIXA:RLMATRIX;VAR D:RLMATRIX);
(*  Compute dij for each i and j                                          *)
VAR L,I,J:INTEGER;
     A,T:REAL;
BEGIN
    FOR I:=1 TO M DO
       BEGIN
           FOR L:=1 TO NO DO
             BEGIN
                J:=ORDER[L];
                 IF((OBJROW[J]<0) AND (X[J]=0)) OR ((OBJROW[J]>0)
                      AND (X[J]=1)) THEN
                       D[I,J]:=0
                 ELSE
                    BEGIN
                        T:=OBJROW[J]*MATRIXA[I,J];
                        IF (T>0) THEN
                          BEGIN
                              IF (MATRIXA[I,J] <0) THEN
                                   A:=-1*MATRIXA[I,J]
                              ELSE
                                   A:=MATRIXA[I,J];
                             D[I,J]:=S[I]/A;
                          END;
                        IF ((T<0) OR (MATRIXA[I,J]=0)) THEN
                             D[I,J]:=10000000;
                    END;
               END;
        END;
END;


(*  Step 2 of Part 2 of Phase 3                                           *)

PROCEDURE PART22(NO:INTEGER;ORDER:INTCOLS;VAR NEWD:INTCOLS;D:RLMATRIX)
(*  Compute dj for each j                                                 *)
VAR L,I,J,MIN:INTEGER;
BEGIN
    FOR L:=1 TO NO DO
      BEGIN
          MIN:=1000;
          J:=ORDER[L];
         FOR I:=1 TO M DO
           BEGIN
               IF (D[I,J] >=0) THEN
                NEWD[J]:=TRUNC(D[I,J])
               ELSE
                 NEWD[J]:=TRUNC(D[I,J] -1);
                IF (NEWD[J]<MIN) THEN
                    MIN:=NEWD[J];
           END;
          NEWD[J]:=MIN;
      END;
END;


(*  Step 3 of Part 2 of Phase 3                                           *)
```

```
PROCEDURE PART22(NO:INTEGER;VAR AR:COLS;ORDER,TEMPC,NEWD:INTCOLS);
(*  Compute Rj for variables with non-zero objective row coefficients
VAR L,J:INTEGER;
BEGIN
    FOR L:=1 TO NO DO
      BEGIN
        J:=ORDER[L];
       AR[J]:=TEMPC[J]*NEWD[J];
      END;
END;


(*  Step 5 of Part 2 of Phase 3                                          *)

PROCEDURE PART25(VAR TEMPC:INTCOLS;K:INTEGER;VAR S:ROWS;OBJROW:INTCOL
      MATRIXA:RLMATRIX;VAR X:INTCOLS);
(*  Check the sign of Ck                                                 *)
VAR I:INTEGER;
BEGIN
    IF (OBJROW[K] > 0) AND NOT(X[K]=1) THEN
      BEGIN
          X[K]:=X[K] + 1;
          FOR I:=1 TO M DO
            S[I]:=S[I]-MATRIXA[I,K];
      END
      ELSE
    IF (OBJROW[K] < 0) AND (X[K]=1)  THEN
      BEGIN
          X[K]:=X[K]-1;
          FOR I:=1 TO M DO
              S[I]:=S[I]+MATRIXA[I,K];
      END
      ELSE
       TEMPC[K]:=0;
END;


(* Step 4 of Part 2 of Phase 3                                           *)

PROCEDURE PART24(OBJROW,ORDER:INTCOLS;MATRIXA:RLMATRIX;VAR S:ROWS;
        VAR X,TEMPC:INTCOLS;NO:INTEGER;AR:COLS;VAR K:INTEGER;
         VAR ENDPART2:BOOLEAN);
(*  Find the maximum r and set k to the index of the maximum r          *)
VAR MAX:REAL;
    L,J:INTEGER;
BEGIN
    MAX:=-1000000;
    FOR L:=1 TO NO DO
      BEGIN
         J:=ORDER[L];
         IF (AR[J] > MAX) THEN
           BEGIN
              MAX:=AR[J];
              K:=J;
           END;
      END;
    IF (AR[K] >0) THEN
          PART25(TEMPC,K,S,OBJROW,MATRIXA,X)
     ELSE
       ENDPART2:=TRUE;
```

```
END;


(*  Part 3 of Phase 3                                                    *)

PROCEDURE PART3(NO:INTEGER;ORDER,OBJROW:INTCOLS;VAR NEWR,RPRIME:MATRIX
(*  Compute Rjk and R'jk                                                 *)
VAR L,J,K,M:INTEGER;
DIVISION:REAL;
BEGIN
    FOR J:=1 TO NO-1 DO
      BEGIN
           L:=ORDER[J];
          FOR K:=J+1 TO NO DO
            BEGIN
                  M:=ORDER[K];
                    DIVISION:=OBJROW[L]/OBJROW[M];
                IF (DIVISION<0) THEN
                   DIVISION:=DIVISION*(-1);
                NEWR[L,M]:=TRUNC(DIVISION-0.00000001);
                RPRIME[L,M]:=TRUNC(DIVISION+1);
            END;
       END;
END;


(*  STep 2 of Part 4 of Phase 3                                          *)

PROCEDURE PART42(J:INTEGER;VAR SPRIME,S:ROWS;DELTA:INTCOLS;
     MATRIXA:RLMATRIX;VAR Q:INTCOLS);
(*  Compute s'                                                           *)
VAR P,I,L:INTEGER;
BEGIN
    L:=0;
    FOR I:=1 TO M DO
      BEGIN
           SPRIME[I]:=S[I]-(DELTA[J]*MATRIXA[I,J]);
           IF (SPRIME[I]<0) THEN
             BEGIN
                  L:=L+1;
                  Q[L]:=I;
             END;
      END;
END;


(*  Step 1 of Part 4 of Phase 3                                          *)

PROCEDURE PART41(VAR SPRIME,S:ROWS;MATRIXA:RLMATRIX;VAR Q:INTCOLS;
    J:INTEGER;X,DELTA:INTCOLS;VAR ENDPART4,INVESTIGATE:BOOLEAN);
(*  Check the sign of (xj + deltaj)                                      *)
BEGIN
    IF (X[J]+DELTA[J] >=0) AND (X[J]+DELTA[J]<=1) THEN
       BEGIN
           INVESTIGATE:=TRUE;
         PART42(J,SPRIME,S,DELTA,MATRIXA,Q)
       END
    ELSE
       BEGIN
       ENDPART4:=TRUE;
```

```
          INVESTIGATE:=FALSE;
         END;
END;


(*  Step 3 of Part 4 of Phase 3                                          *)

PROCEDURE PART43(DELTA,Q:INTCOLS;VAR X:INTCOLS;VAR S:ROWS;SPRIME:ROWS;
         VAR ENDPART4,INVESTIGATE:BOOLEAN);
(*  Check if Q = 0                                                      *)
VAR I:INTEGER;
BEGIN
    IF (Q[1]=0) THEN
       BEGIN
            X[J]:=X[J]-DELTA[J];
            FOR I:=1 TO M DO
              S[I]:=SPRIME[I];
           INVESTIGATE:=FALSE;
         END
      ELSE
         BEGIN
             ENDPART4:=TRUE;
             INVESTIGATE:=TRUE;
         END;
END;


(*  Step 1 of Part 5 of Phase 3                                         *)

PROCEDURE PART51(J,K:INTEGER;NEWP:MATRIX;X,OBJROW:INTCOLS;
           VAR L:INTROWS);
(*  Check the sign of Ck                                                 *)
BEGIN
    IF (OBJROW[K] > 0) THEN
      BEGIN
          IF (X[K]<NEWP[J,K]) THEN
             L[K]:=-X[K]
          ELSE
             L[K]:=-NEWP[J,K];
      END
     ELSE
        L[K]:=-X[K];
END;


(*  Step 2 of Part 5 of Phase 3                                          *)

PROCEDURE PART52(P:INTROWS;K:INTEGER;VAR U:INTROWS;SPRIME:ROWS;
        MATRIXA:PLMATRIX);
(*  Compute Uk                                                            *)
VAR
   I,J,L,T:INTEGER;
    R:REAL;
BEGIN
    S:=10000000;
    FOR L:=1 TO M DO
      BEGIN
          I:=P[L];
          IF NOT(I=0) THEN
            BEGIN
```

```
                IF NOT(MATRIXA[I,K]=0) THEN
                    BEGIN
                      R:= SPRIME[I]/MATRIXA[I,K];
                      IF (R>=0) THEN
                         T:=TRUNC(R)
                      ELSE
                         T:=TRUNC(R-1);
                   END

                ELSE
                   T:=10000000;
                IF (T<S) THEN
                   S:=T;
            END;
       END;
     U[K]:=S;
END;


(*  Step 4 of Part 5 of Phase 3                                     *)

PROCEDURE PARTS4(MATRIXA:PLMATRIX;K:INTEGER;S:ROWS;
        VAR L,LPRIME:INTROWS);
(*  Compute Lk and L'k                                              *)
VAR MAX,I,COUNT,V:INTEGER;
      T:REAL;
BEGIN
    COUNT:=0;
    MAX:=-10000000;
    FOR I:=1 TO M DO
      BEGIN
          IF (MATRIXA[I,K] < 0) THEN
            BEGIN
                T:=(S[I]/MATRIXA[I,K]);
                  IF (T < 0) THEN
                    V:=TRUNC(T)
                  ELSE
                    V:=TRUNC(T+0.99999999);
                IF (V>MAX) THEN
                    MAX:=V;
            END
          ELSE
             COUNT:=COUNT+1;
      END;
     IF (COUNT=M) THEN
        LPRIME[K]:=-10000000
        ELSE
     LPRIME[K]:=MAX;
     IF (LPRIME[K]>L[K]) THEN
        L[K]:=LPRIME[K];
END;


(*  Step 3 of Part 5 of Phase 3                                     *)

PROCEDURE PARTS3(MATRIXA:PLMATRIX;S:ROWS;VAR L,LPRIME:INTROWS;
          K:INTEGER;U:INTROWS;VAR ENDPARTS:BOOLEAN);
(*  Check if Lk <= Uk                                               *)
BEGIN
    IF (L[K]<=U[K]) THEN
```

```
            PART54(MATRIXA,K,S,L,LPRIME)
        ELSE
            ENDPART5:=TRUE;
END;


(*  Step 6 of Part 5 of Phase 3                                    *

PROCEDURE PART56(J,K:INTEGER;VAR X:INTCOLS;VAR S:ROWS;
          OBJROW,DELTA:INTCOLS;L:INTROWS;SPRIME:ROWS;U:INTROWS;
          MATRIXA:FLMATRIX;VAR ENDPART5:BOOLEAN;VAR Z:INTEGER);
(*  Check the sign of Ck in order to select the improved solution   *)
VAR G,I:INTEGER;
BEGIN
    IF (OBJROW[K]>0) AND (X[K]<1) THEN
      BEGIN
           X[J]:=X[J]+DELTA[J];
           X[K]:=X[K]+U[K];
           FOR I:=1 TO M DO
             S[I]:=SPRIME[I]-(U[K]*MATRIXA[I,K]);
           ENDPART5:=TRUE;
      END
    ELSE
       IF (OBJROW[K]<=0)  THEN
       BEGIN
            X[J]:=X[J]+DELTA[J];
            X[K]:=X[K]+L[K];
            FOR I:=1 TO M DO
              S[I]:=SPRIME[I]-(L[K]*MATRIXA[I,K]);
            ENDPART5:=TRUE;
       END;
    Z:=0;
    FOR G:=1 TO N DO
       Z:=Z+(X[G]*OBJROW[G]);
END;


(*  Step 5 of Part 5 of Phase 3                                       *

PROCEDURE PART55(VAR X:INTCOLS;J,K:INTEGER;VAR S,SPRIME:ROWS;
   OBJROW,DELTA:INTCOLS;MATRIXA:FLMATRIX;VAR Z:INTEGER;L,U:INTROWS;
   VAR ENDPART5:BOOLEAN);
(*  Check if Lk <= Uk                                                *)
BEGIN
    IF (L[K]<=U[K]) THEN
       PART56(J,K,X,S,OBJROW,DELTA,L,SPRIME,U,MATRIXA,ENDPART5,Z)
    ELSE
       ENDPART5:=TRUE;
END;


(*  Step 1 of Part 6 of Phase 3                                      *)


PROCEDURE PART61(J,K:INTEGER;OBJROW:INTCOLS;VAR U:INTROWS;NEWR:MATRIX)
(*  Check the sign of Ck                                             *)
BEGIN
    IF (OBJROW[K]<0) THEN
          U[K]:=NEWR[J,K]
    ELSE
```

```
       U[K]:=10000000;
END;


(*  Step 2 of Part 6 of Phase 3                                        *)

PROCEDURE PART62(SPRIME:ROWS;MATRIXA:RLMATRIX;K:INTEGER;
      VAR L:INTROWS;Q:INTROWS);
(*  Compute Lk                                                          *
VAR Z,MAX,T,I:INTEGER;
V:REAL;
BEGIN
    MAX:=-10000000;
    FOR I:=1 TO M DO
      BEGIN
          T:=Q[I];
          IF NOT(T=0) THEN
            BEGIN
                V:=(SPRIME[T]/MATRIXA[T,K]);
                  IF (V<0) THEN
                    Z:=TRUNC(V)
                  ELSE
                    Z:=TRUNC(V+0.9999999);
                IF (Z>MAX) THEN
                      MAX:=Z;
            END;
      END;
    L[K]:=MAX;
END;



(*  Step 4 of Part 6 of Phase 3                                         *
PROCEDURE PART64(K:INTEGER;SPRIME:ROWS;MATRIXA:RLMATRIX;
      VAR U,UPRIME:INTROWS);
(*  Compute Uk and U'k                                                 *)
VAR MIN,V,COUNT,I:INTEGER;
  T:REAL;
BEGIN
    MIN:=10000000;
    FOR I:=1 TO M DO
      BEGIN
          IF (MATRIXA[I,K] >0) THEN
            BEGIN
                T:=(SPRIME[I]/MATRIXA[I,K]);
                 IF (T>=0) THEN
                    V:=TRUNC(T)
                 ELSE
                    V:=TRUNC(T-1);
                IF (V<MIN) THEN
                   MIN:=V;
                COUNT:=COUNT+1;
            END;
       END;
     IF (COUNT=0) THEN
       UPRIME[K]:=10000000
     ELSE
        UPRIME[K]:=MIN;
     IF (UPRIME[K]<U[K]) THEN
         U[K]:=UPRIME[K];
```

```
END;


(*  Step 3 of Part 6 of Phase 3                                             *)

PROCEDURE PART63(SPRIME:ROWS;MATRIXA:RLMATRIX;VAR U,UPPRIME:INTROWS;
           K:INTEGER;L:INTROWS;VAR ENDPART6:BOOLEAN);
(*  Check if Lk <= Uk                                                      *)
BEGIN
    IF (L[K]<=U[K]) THEN
       PART64(K,SPRIME,MATRIXA,U,UPPRIME)
    ELSE
       ENDPART6:=TRUE;
END;


(*  Step 6 of Part 6 of Phase 3                                           *)

PROCEDURE PART66(OBJROW:INTCOLS;J,K:INTEGER;VAR ENDPART6:BOOLEAN;
       VAR L:INTROWS;VAR S,SPRIME:ROWS;VAR X:INTCOLS;MATRIXA:RLMATRIX;
       DELTA:INTCOLS;U:INTROWS;VAR Z:INTEGER);
(*  Check the sign of Ck in order to select the improved solution *)
VAR G:INTEGER;
BEGIN
    IF (OBJROW[K]>0) AND (X[K] < 1) THEN
      BEGIN
          X[J]:=X[J]+DELTA[J];
          X[K]:=X[K]+U[K];
          FOR I:=1 TO M DO
             S[I]:=SPRIME[I]-(MATRIXA[I,K]);

       END
    ELSE
       IF  (OBJROW[K]<=0)  THEN
         BEGIN
            X[J]:=X[J]+DELTA[J];
            X[K]:=X[K]+L[K];
            FOR I:=1  TO M DO
               S[I]:=SPRIME[I]-(L[K]*MATRIXA[I,K]);
         END;
    ENDPART6:=TRUE;
    Z:=0;
    FOR G:=1 TO N DO
      Z:=Z+(X[G]*OBJROW[G]);
END;


(*  Step 5 of Part 6 of Phase 3                                             *)

PROCEDURE PART65(J,K:INTEGER;OBJROW,DELTA:INTCOLS;VAR L:INTROWS;
         VAR S,SPRIME:ROWS;VAR X:INTCOLS;MATRIXA:RLMATRIX;VAR Z:INTEGE
         U:INTROWS;VAR ENDPART6:BOOLEAN);
(*  Check if Lk <= Uk                                                      *)
BEGIN
    IF (L[K]<=U[K]) THEN
       PART66(OBJROW,J,K,ENDPART6,L,S,SPRIME,X,MATRIXA,DELTA,U,Z)
    ELSE
       ENDPART6:=TRUE;
END;
```

```
(*  Part 1 of Phase 3                                                     *)

PROCEDURE PART1(XP:INTCOLS;VAR ORDER,OBJROW,TEMPC,DELTA:INTCOLS;
          VAR NO:INTEGER;VAR S:ROWS;RHS:ROWS;MATRIXA:RLMATRIX;X:INTCOLS);
VAR P:INTEGER;
BEGIN
     PART11(X,XP);
     PART12(ORDER,OBJROW,TEMPC,DELTA,NO);
     FOR P:=1 TO N DO
       WRITE(OUTFILE,X[P]);
     WRITELN(OUTFILE);
     PART13(S,RHS,MATRIXA,X);
END;


(*  Part 2 of Phase 3                                                    *)

PROCEDURE PART2(K:INTEGER;NO:INTEGER;VAR S:ROWS;ORDER,OBJROW:INTCOLS;
       VAR X:INTCOLS;MATRIXA:RLMATRIX;VAR D:RLMATRIX;VAR NEWD:INTCOLS;
       VAR AR:COLS;VAR TEMPC:INTCOLS;VAR ENDPART2:BOOLEAN);

BEGIN
     WHILE NOT(ENDPART2) DO
          BEGIN
               PART21(NO,S,ORDER,OBJROW,X,MATRIXA,D);
               PART22(NO,ORDER,NEWD,D);
               PART23(NO,AR,ORDER,TEMPC,NEWD);
               PART24(OBJROW,ORDER,MATRIXA,S,X,TEMPC,NO,AR,K,ENDPART2);
          END;
     RESULTS(X,ELIGIBLE,OBJROW,I,2);
END;


(*  Part 4 of Phase 3                                                     *)

PROCEDURE PART4(J:INTEGER;VAR SPRIME,S:ROWS;VAR X,DELTA:INTCOLS;
      MATRIXA:RLMATRIX;VAR Q:INTCOLS;VAR ENDPART4,INVESTIGATE:BOOLEAN);
VAR P:INTEGER;
BEGIN
     PART41(SPRIME,S,MATRIXA,Q,J,X,DELTA,ENDPART4,INVESTIGATE);
     IF NOT (ENDPART4) THEN
       PART43(DELTA,Q,X,S,SPRIME,ENDPART4,INVESTIGATE);
     RESULTS(X,ELIGIBLE,OBJROW,I,4);
END;


(* Step 9 of Phase 3 when different parts are fitted together          *)

PROCEDURE CHECKSTPP(ORDER:INTCOLS;NO:INTEGER;VAR A,T,J,K:INTEGER;
        VAR CHECKJ:BOOLEAN);
(*  Check if j < min (n0,n-1)                                              *
VAR MIN:INTEGER;
BEGIN
     IF (NO<N-1) THEN
       MIN:=NO
     ELSE
       MIN:=N-1;
     IF (A < MIN ) THEN
        BEGIN
```

```
            A:=A+1;
            T:=N;
            J:=ORDER[A];
             K:=ORDER[T];
        END
      ELSE
         CHECKJ:=FALSE;
END;


(*  Step 10 of Phase 3 when different parts are fitted together      *

PROCEDURE CHECKSTP10(VAR SAME:BOOLEAN;X,XL:INTCOLS);
(*  Check if x is not equal to xl                                    *
VAR G:INTEGER;
BEGIN
    SAME:=TRUE;
    FOR G:=1 TO N DO
      IF NOT(X[G]=XL[G]) THEN
          SAME:=FALSE;
END;


(*  Reset the value of S' in Step 1 of Part 7                       *

PROCEDURE RESET1(J:INTEGER;VAR SPRIME:ROWS;S:ROWS;DELTA:INTCOLS;
     MATRIXA:RLMATRIX);

(*  Set S' = Si + deltaj . Ai,j                                      *
VAR I:INTEGER;
BEGIN
    FOR I:=1 TO M DO
      SPRIME[I]:=S[I]+DELTA[J]*MATRIXA[I,J];
END;


(*  Reset the values of x and S at Step 5 of Part 7                  *)

PROCEDURE RESET5(J,K:INTEGER;VAR X:INTCOLS;VAR S:ROWS;SPRIME:ROWS;
     DELTA:INTCOLS;RPRIME:MATRIX;MATRIXA:RLMATRIX);

VAR I:INTEGER;
BEGIN
    X[J]:=X[J]-DELTA[J];
    X[K]:=X[K]-(DELTA[K]*RPRIME[J,K]);
    FOR I:=1 TO M DO
       S[I]:=SPRIME[I]-DELTA[J]*RPRIME[J,K]*MATRIXA[I,K];
END;


(*  Check the sign of (S' - deltak . Rj,k . Ai,k ) in Step4 of Part 7

PROCEDURE CHECKSLACK(J,K:INTEGER;VAR INFEAS:BOOLEAN;SPRIME:ROWS;
      DELTA:INTCOLS;MATRIXA:RLMATRIX;RPRIME:MATRIX);

VAR I:INTEGER;
    T:REAL;
BEGIN
    INFEAS:=FALSE;
    FOR I:=1 TO M DO
```

```
          BEGIN
               T:=SPRIME[I]-DELTA[K]*RPRIME[J,K]*MATRIXA[I,K];
               IF (T<0) THEN
                  INFEAS:=TRUE;
          END;
     END;


     (*   Part 5 of Phase 3                                                  *

     PROCEDURE PART5(Z:INTROWS;J,K:INTEGER;NEWR:MATRIX;VAR X:INTCOLS;
          OBJROW,DELTA:INTCOLS;VAR L,LPRIME,U:INTROWS;VAR S:ROWS;
          SPRIME:ROWS;MATRIXA:RLMATRIX;VAR ENDPART5:BOOLEAN);
     BEGIN
        PART51(J,K,NEWR,X,OBJROW,L);
        PART52(Z,K,U,SPRIME,MATRIXA);
        PART53(MATRIXA,S,L,LPRIME,K,U,ENDPART5);
        IF NOT(ENDPART5) THEN
              PART55(X,J,K,S,SPRIME,OBJROW,DELTA,MATRIXA,Z,L,U,ENDPART5);
        RESULTS(X,ELIGIBLE,OBJROW,Z,5);
     END;


     (*   Part 6 of Phase 3

     PROCEDURE PART6(J,K:INTEGER;DELTA,OBJROW:INTCOLS;VAR L,U,UPRIME:INTROW
          ;NEWR:MATRIX;Z:INTROWS;VAR S,SPRIME:ROWS;MATRIXA:RLMATRIX;
          VAR ENDPART6:BOOLEAN;VAR X:INTCOLS;VAR Z:INTEGER);
     BEGIN
        PART61(J,K,OBJROW,U,NEWR);
        PART62(SPRIME,MATRIXA,K,L,Z);
        PART63(SPRIME,MATRIXA,U,UPRIME,K,L,ENDPART6);
        IF NOT(ENDPART6) THEN
          PART65(J,K,OBJROW,DELTA,L,S,SPRIME,X,MATRIXA,Z,U,ENDPART6);
        RESULTS(X,ELIGIBLE,OBJROW,Z,6);
     END;


     (*   Part 7 of Phase 3                                                 *)

     PROCEDURE PART7(ORDER:INTCOLS;VAR INFEAS:BOOLEAN;VAR A,T,K:INTEGER;
        J:INTEGER;VAR X:INTCOLS;VAR SPRIME,S:ROWS;DELTA:INTCOLS;
        RPRIME:MATRIX;MATRIXA:RLMATRIX;VAR ENDPART7:BOOLEAN;NO:INTEGER);

     VAR    STEP2:BOOLEAN;

     BEGIN
        ENDPART7:=FALSE;
              IF ((X[J]-DELTA[J])>=0) THEN
                BEGIN
                     RESET1(J,SPRIME,S,DELTA,MATRIXA);
                     STEP2:=TRUE;
                END
              ELSE
                ENDPART7:=TRUE;
              WHILE (STEP2) AND NOT(ENDPART7)  DO
                  BEGIN
                     IF (T>NO) THEN
                        ENDPART7:=TRUE
                     ELSE
                     BEGIN
                         IF ((X[K]+DELTA[K]*RPRIME[J,K])>=0) AND ((X[K]-
                           DELTA[K]*RPRIME[J,K])<=1) THEN
                           BEGIN
                               STEP2:=FALSE;
                               CHECKSLACK(J,K,INFEAS,SPRIME,DELTA,MATRI
                               IF NOT(INFEAS) THEN
                                 BEGIN
                                     STEP2:=FALSE;
                                     RESETS(J,K,X,S,SPRIME,DELTA,RPRIME
                                  END
                                ELSE
                                 BEGIN
                                    T:=T+1;
                                    K:=ORDER[T];
                                 END;
                            END
                          ELSE
                           BEGIN
                            T:=T+1;
                             K:=ORDER[T];
                           END;
                    END;
          END;
    RESULTS(X,ELIGIBLE,OBJROW,Z,T);
END;
(*              MAIN PROGRAM                                               *)


(*  Steps 4,5 and 6 of Phase I,when different parts are fitted togethe

PROCEDURE CHECKSTP4(J,I,K:INTEGER;VAR ENDPARTS,ENDPART6,IMPROVED:
    BOOLEAN;VAR STP5,STP6,STP7:BOOLEAN;NEWR:MATRIX;VAR L,LPRIME,U,
     UPRIME:INTROWS;VAR S,SPRIME:ROWS;Q:INTROWS;MATRIXA:RLMATRIX;
     OBJROW,DELTA:INTCOLS;VAR X:INTCOLS);
VAR G,ZVAL,F,I:INTEGER;
BEGIN
    FOR F:=1 TO M DO
       BEGIN
(*  Check the sign of Ai,k for i an element of Q                           *)
(*  if Ai,k > 0 for every such i, then go to step 5                          *
(*  if Ai,k < 0 for every such i, then go to step 6                          *
(*  if neither then go to step 7                                             *
           I:=Q[F];
            IF NOT(I=0) THEN
               BEGIN
                    IF (MATRIXA[I,K]<=0) THEN
                         STP5:=FALSE;
                    IF (MATRIXA[I,K]>=0) THEN
                         STP6:=FALSE;
               END;
       END;
        IF (STP5 OR STP6) THEN
           STP7:=FALSE
       ELSE
          STP7:=TRUE;
       IF (STP5) THEN
         PARTS(I,J,K,NEWR,X,OBJROW,DELTA,L,LPRIME,U,S,SPRIME,MATRIXA,E
        ELSE
           IF (STP6) THEN
```

```
(* Check if an improved solution is found                                      *
              PART6(J,K,DELTA,OBJROW,L,U,UPRIME,NEWR,Q,S,SPRIME,MATRIXA
          IF NOT(STP7) THEN
              BEGIN
                   ZVAL:=0;
                   FOR G:=1 TO N DO
                     ZVAL:=ZVAL+(X[G]*OBJROW[G]);
                   IF (ZVAL <= Z) THEN
                        IMPROVED:=FALSE
                   ELSE
                      IMPROVED:=TRUE;
              END;
END;


(*  Initialize all the variables                                                *

PROCEDURE INITIALIZE(VAR COUNTER:INTEGER;VAR POSSIBLE,FOUND,TERMINATE,
    ENDPHASE2,STP10,INVESTIGATE,ENDPART2,ENDPART3,ENDPART4,ENDPART5,
    ENDPART6,ENDPART7,SAME,IMPROVED,STP5,STP6,CHECKJ:BOOLEAN;
    VAR NUM:INTEGER;VAR Q:INTROWS;VAR ORDER,LET2:INTCOLS);
VAR I,J:INTEGER;
BEGIN
    ENDPART2:=FALSE;
    ENDPART3:=FALSE;
    ENDPART4:=FALSE;
    ENDPART5:=FALSE;
    ENDPART6:=FALSE;
    ENDPART7:=FALSE;
    SAME:=FALSE;
    IMPROVED:=FALSE;
    FOR I:=1 TO M DO
      Q[I]:=0;
    CHECKJ:=TRUE;
    STP5:=TRUE;
    STP6:=TRUE;
    INVESTIGATE:=TRUE;
    FOR J:=1 TO N DO
      BEGIN
       LET2[J]:=0;
       ORDER[J]:=J;
      END;
    FOUND:=FALSE;
    TERMINATE:=FALSE;
    NUM:=1;
    ENDPHASE2:=FALSE;
    POSSIBLE:=TRUE;
    COUNTER:=0;
    STP10:=FALSE;
 END;


 (* Normalize the coefficients of the problem                                  *)

 PROCEDURE NORMALIZE(VAR MATRIXA:PLMATRIX;VAR RHS:ROWS);
 VAR I,J,K:INTEGER;
    SUM:REAL;
 BEGIN
     FOR I:=1 TO M DO
       BEGIN
```

```
                SUM:=0;
                FOR K:=1 TO N DO
                   BEGIN
                       SUM:=SQR(MATRIXA[I,K])+SUM;
                   END;
                 SUM:=SQRT(SUM);
             FOR J:=1 TO N DO
                MATRIXA[I,J]:=MATRIXA[I,J] / SUM;
               RHS[I]:=RHS[I]/SUM;
        END;
END;



BEGIN
    RESET(INFILE);
    REWRITE(OUTFILE);
(*  Read the problem                                                  *)
    READPROB(CRITERION,MATRIXA,OBJROW,M,N,RHS,OPGRHS);
(*  Read the results from Phase 1                                     *)
    READX1(ZLIN,X1);
    READBFSOLN(XBF,TOTLINES);
(*  Initialize                                                        *)
    INITIALIZE(COUNTER,POSSIBLE,FOUND,TERMINATE,ENDPHASE2,STP10,
       INVESTIGATE,ENDPART2,ENDPART3,ENDPART4,ENDPART5,ENDPART6,
       ENDPART7,SAME,IMPROVED,STP5,STP6,CHECKJ,NUM,Q,ORDER,LETQ);
(*  Normalize                                                         *)
    NORMALIZE(MATRIXA,RHS);
(*  Phase 2.In this Phase, one tries to find a feasible solution     *)
(*  on the line segment (or segments) between x1 and x2              *)
    STEP1(ALFA);
    WHILE NOT(FOUND) AND (POSSIBLE) DO
      BEGIN
          ADJUST(TOTLINES,X1,XBF,NUM,X2);
    WHILE NOT(TERMINATE) AND NOT(ENDPHASE2) DO
      BEGIN
        STEP2(TEMPX,X1,X2,ALFA);
        STEP3(TEMPX,X);
        STEP4(SUMQ,QROWS,MATRIXA,X);
        WHILE NOT(STP10) AND NOT(ENDPHASE2) DO
          BEGIN
            STEP5(NEWQ,QROWS,DELX,XF,X,MATRIXA,QSTAR,SUMQ,FOUND,ENDP
            IF NOT(ENDPHASE2) THEN
              BEGIN
                STEP7(STP10,ALFA,QSTAR,SUMQ,LETQ);
                  IF NOT(STP10) THEN
                       STEP8(OBJROW,CRITERION,QSTAR,NEWQ,DELX,X,OPG
              END;
          END;
      IF NOT(FOUND) THEN
        BEGIN
         STEP10(X1,X2,ALFA,STP10);
         STEP11(ALFA,TERMINATE);
        END;
    END;
    IF NOT(FOUND) THEN
      BEGIN
          FOR R:=1 TO N DO
              X1[R]:=X[R];
          IF NUM < TOTLINES THEN
```

```
              NUM:=NUM+1
           ELSE  POSSIBLE:=FALSE;
        END;
     END;
     CONT:=TRUE;
     RESULTS(X,ELIGIBLE,OBJROW,Z,1);
(*   Phase 3.In this Phase, one tries to improve the solution found *)
(*   in Phase 2. Two alternating modes and Phase 2 type search are  *)
(*   used for this.                                                  *)
     PART1(XF,ORDER,OBJROW,TEMPC,DELTA,NO,S,RHS,MATRIXA,X);
     PART3(NO,ORDER,OBJROW,NEWR,PPRIME);
     WHILE NOT(SAME) DO
        BEGIN
(* First mode                                                        *)
                  PART2(K,NO,S,ORDER,OBJROW,X,MATRIXA,D,NEWD,AR,TEMPC,ENDPARTI
                  FOR E:=1 TO N DO
                       XL[E]:=X[E];
                  A:=1;
                  T:=N;
                  J:=ORDER[A];
                  K:=ORDER[T];
                  WHILE (CHECKJ) DO
                      BEGIN
(*   Second mode                                                     *)
                          PART4(J,SPRIME,S,X,DELTA,MATRIXA,Q,ENDPART4,INVESTI

                                IF (INVESTIGATE) THEN
                                  BEGIN
                                    WHILE (CONT) AND (NOT(IMPROVED)) DO
                                      BEGIN
                                        CHECKSTP4(J,Z,K,ENDPART5,ENDPART6,
,Q,MATRIXA,OBJROW,DELTA,X);
                                        IF (NOT(IMPROVED)) OR (STP7) THEN
                                           BEGIN
                                             IF ((T-1) > A) THEN
                                               BEGIN
                                              T:=T-1;
                                               K:=ORDER[T];
                                                END
                                               ELSE
                                                 CONT:=FALSE;
                                            END
                                      END;
                                  END
                                ELSE
                                 BEGIN
                                   T:=A+1;
                                   K:=ORDER[T];
                                 END;
                                  PART7(ORDER,INFEAS,A,T,K,J,X,SPRIME,S,DELT
                                  CHECKSTP9(ORDER,NO,A,T,J,K,CHECKJ);
                      END;
                                  CHECKSTP10(SAME,X,XL);
        END;
      RESULTS(Y,ELIGIBLE,OBJROW,Z,8);
     M:=M+1;
     FOUND:=FALSE;
     FOR F:=1 TO N DO
      BEGIN
        YF[F]:=Y[F];
```

```
              MATRIXA[M,P]:=-OBJROW[P];
              SUM:=(OBJROW[P]*X[P])+SUM;
         END;
(*   Phase 2 type  search of Method 5 of Phase 3                    *)
(*   Add the new constraint                                         *)
     RHS[M]:=-(SUM+1);
     FOR P:=1 TO N DO
      SQTERM:=SQTERM+SQR(MATRIXA[M,P]);
     TERM:=SQRT(SQTERM);
     FOR P:=1 TO N DO
       MATRIXA[M,P]:=MATRIXA[M,P] / TERM;
     RHS[M]:=RHS[M] / TERM;
     STEP4(SUMQ,QROWS,MATRIXA,X);
     WHILE NOT (FOUND) AND (COUNTER < 100) DO
       BEGIN
          STEP5(NEWQ,QROWS,DELX,XF,X,MATRIXA,QSTAR,SUMQ,FOUND,ENDPHASE
          STEP72(OBJROW,CRITERION,K,SUMQ,QSTAR,DELX);
          STEP9(K,X,DELX,QROWS,NEWQ,QSTAR,SUMQ);
          COUNTER:=COUNTER+1;
       END;
     RESULTS(XF,ELIGIBLE,OBJROW,I,9);
(*   Compute the square root of the sum of the square of Cj         *)
(*   for j=1..n                                                     *)
     DEV:=0;
     FOR P:=1 TO N DO
       DEV:=DEV+SQR(OBJROW[P]);
     DEV:=SQRT(DEV);
     WRITELN(OUTFILE,DEV);
END.
```

REFERENCES

1. Balas, E., "An Additive Algorithm for Solving Linear Programs with Zero-One Variables," Operations Research, Vol. 13, No. 4 (1965), pp. 517-546.

2. Balas, E., and Martin, C.H., "Pivot and Complement--A heuristic for 0-1 Programming," Management Science, Vol. 26, No. 1 (1980), pp. 86-96.

3. Bender, J.F., "Partitioning Procedures for Solving Mixed-Variables Programming Problems," Numerische Mathematik, Vol. 4 (1962), pp. 238-262.

4. Crowder, H., Johnson, E.L., and Padberg, M., "Solving Large-Scale Zero-One Linear Programming Problems," Operations Research, Vol. 31, No. 5 (1983), pp. 803-834.

5. Dakin, R.J., "A Tree Search Algorithm for Mixed Integer Programming Problems," Computer Journal, Vol. 8, No. 3 (1965), pp. 250-255.

6. Echols, R.E. and Cooper, L., "Solution of Integer Linear Programming Problems by Direct Search," J. Assoc. Comput. Mach., Vol. 15 (1968), pp. 75-84.

7. Faaland, B.H. and Hillier, F.S., "Interior Path Methods for Heuristic Integer Programming Procedures," Operations Research, Vol. 27, No. 6 (1979), pp. 1069-1087.

8. Geoffrion, A.M., "Integer Programming by Implicit Enumeration and Balas' Method," SIAM Review, Vol. 9, No. 2 (April 1967), pp. 178-190.

REFERENCES

1. Balas, E., "An Additive Algorithm for Solving Linear Programs with Zero-One Variables," Operations Research, Vol. 13, No. 4 (1965), pp. 517-546.

2. Balas, E., and Martin, C.H., "Pivot and Complement--A heuristic for 0-1 Programming," Management Science, Vol. 26, No. 1 (1980), pp. 86-96.

3. Bendas, J.F., "Partitioning Procedures for Solving Mixed-Variables Programming Problems," Numerische Mathematik, Vol. 4 (1962), pp. 238-262.

4. Crowder, H., Johnson, E.L., and Padberg, M., "Solving Large-Scale Zero-One Linear Programming Problems," Operations Research, Vol. 31, No. 5 (1983), pp. 803-834.

5. Dakin, R.J., "A Tree Search Algorithm for Mixed Integer Programming Problems," Computer Journal, Vol. 8, No. 3 (1965), pp. 250-255.

6. Echols, R.E. and Cooper, L., "Solution of Integer Linear Programming Problems by Direct Search," J. Assoc. Comput. Mach., Vol. 15 (1968), pp. 75-84.

7. Faaland, B.H. and Hillier, F.S., "Interior Path Methods for Heuristic Integer Programming Procedures," Operations Research, Vol. 27, No. 6 (1979), pp. 1069-1087.

8. Geoffrion, A.M., "Integer Programming by Implicit Enumeration and Balas' Method," SIAM Review, Vol. 9, No. 2 (April 1967), pp. 178-190.

9. Geoffrion, A.M. and Marsten, R.E., "Integer Programming Algorithms: A Framework and State-of-the-Art Survey," Management Science, Vol 18, No. 9 (1972), pp. 465-491.

20. Glover, F., "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem," Operations Research, Vol. 13, No. 6 (1965), pp. 879-919.

11. Gomory, R.E., "All-Integer Programming Algorithm," in J.F. Muth and G.L. Thompson (eds.), Industrial Scheduling, Prentice-Hall, New York, 1963, 193-206. First issued in 1960.

12. ______________, "An algorithm for Integer Solutions to Linear Programs," in R.L. Graves and P. Wolfe (eds.), Recent Advances in Mathematical Programming, McGraw-Hill, New York, 1963, pp. 269-302. First issued in 1958.

13. ______________, "On the Relation between Integer and Non-Integer Solutions to Linear Programs," Proc. Nat. Acad. Sci., Vol. 53 (1965), pp. 260-265.

14. Haldi, J., "25 Integer Programming Test Problems," Working Paper No. 43.

15. Hammer, P.L. and Rudean, S., Boolean Methods in Operations Research and Related Areas, Springer-Verlag, Berlin, 1968.

16. Hillier, F.S., "Efficient Heuristic Procedures for Integer Linear Programming with an Interior," Operations Research, Vol. 17 (1969), pp. 600-637.

17. ______________, "A Bound-and-Scan Algorithm for Pure Integer Linear Programming with General Variables," Technical Report No. 3 (1969), Department of Operations Research, Stanford University.

18. ____________, "A Further Investigation of Efficient Heuristic Procedures for Integer Linear Programming with an Interior," Technical Report, Department of Operations Research, Stanford University, February, 1977.

19. Ibaraki, T., Ohashi, T., and Mine, H. "A Heuristic Algorithm for Mixed Integer Programming Problems," Mathematical Programming, Study 2 (1976), pp. 115-136.

20. Kochenberger, G.A., McCarl, B.A., and Wyman, F.P., "A Heuristic for General Integer Programming," Decision Science, Vol. 5 (1974), pp. 36-44.

21. Land, A.H. and Doig, A.G., "An Automatic Method of Solving Discrete Programming Problems," Econometrica, Vol. 28 (1960), pp. 497-520.

22. Lemke, C.E. and Spielberg, K., "Direct Search Algorithms for Zero-One and Mixed Integer Programming," Operations Research, Vol. 15, No. 5 (1967), pp. 892-914.

23. Petersen, C.C., "Computational Experience with Variants of the Balas Algorithm Applied to the Selection of R&D Projects," Management Science, Vol. 13, No. 9 (1967), pp. 736-750.

24. Rieter, S. and Rice, D.B., "Discrete Optimizing Solution Procedures for Linear and Non-linear Integer Programming Problems," Management Science, Vol. 12 (1966), pp. 829-850.

25. Roth, R.H., "An Approach to Solving Linear Discrete Optimization Problems," J. Assoc. Comput. Mach., Vol. 17 (1970), pp. 300-313.

26. Senju, S. and Toyoda, Y., "An Approach to Linear Programming with 0-1 Variables," Management Sci. Vol. 15 (1968), B196-B207.

27. Shaprio, J.F., "Dynamic Programming Algorithms for the Integer Programming Problem - I: The Integer Programming Problem Viewed as a Knapsack Type Problem." Operations Research, Vol. 16, No. 1 (1968), pp. 103-121.

28. ____________, "Group Theoretic Algorithms for the Integer Programming Problem - II: Extension to a General Algorithm," Operations Research, Vol. 16, No. 5 (1968), pp. 928-947.

29. ____________, "Turnpike Theorems for Integer Programming Problems," Operations Research, Vol. 18, No. 3 (1970), pp. 432-440.

30. Thiriez, H., "Airline Crew Scheduling: A Group Theoretic Approach," Report R-69 (1969), Flight Transportation Laboratory, Massachusetts Institute of Technology.

31. Toyoda, Y., "A Simplified Algorithm for Obtaining Approximate Solutions to Zero-One Programming Problems," Management Science, Vol. 21 (1975), pp. 1417-1427.

32. Trauth, C.A. and Woolsly, R.E., "Integer Linear Programming: A Study in Computational Efficiency," Management Science, Vol. 15, No. 9 (1969), pp. 481-493.

33. Woiler, S., "Implicit Enumerations Algorithms for Discrete Optimization Problems," Technical Report No. 4 (May 1967), Department of Industrial Engineering, Stanford University.

34. Wolsey, L.A., "Mixed Integer Programming: Discretization and the Group Theoretic Approach," Technical Report No. 42 (July 1969), Operations Research Center, Massachusetts Institute of Technology.

35. Zanakis, S.H., "Heuristic 0-1 Linear Programming: An Experimental Comparison of Three Methods," Management Science, Vol. 24, No. 1 (1977), pp. 91-103.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER SOL 87-3	2. GOVT ACCESSION NO. ADA181431	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Heuristic Procedures for 0-1 Integer Programming		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Kadriye A. Ercikan, Frederick S. Hillier		8. CONTRACT OR GRANT NUMBER(s) N00014-85-K-0343
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Operations Research - SOL Stanford University Stanford, CA 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR-047-064
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research - Dept. of the Navy 800 N. Quincy Street Arlington, VA 22217		12. REPORT DATE March 1987
		13. NUMBER OF PAGES 73 pp.
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

integer programming
heuristic procedures
binary variables

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See next page.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

SOL 87-3: Heuristic Procedures for 0-1 Integer Programming,
by Kadriye A. Ercikan, Frederick S. Hillier

The limited success of exact algorithms for solving integer programming problems has encouraged the development of heuristic procedures for efficiently obtaining solutions that are at least close to optimal.

This ~~thesis~~ presents three heuristic procedures for 0-1 integer programming problems having only inequality constraints. These procedures are based on Hillier's previous heuristic procedures for general integer linear programming. All three were successfully run on problems with up to 500 variables with only modest execution times. The quality of the solutions for these problems were, in general, very good and often were optimal. When the best of the solutions obtained by the three procedures was taken, the final solution was optimal for 24 of 45 randomly generated problems.

These procedures can be used for problems that are too large to be computationally feasible for exact algorithms. In addition, they can be useful for smaller problems by quickly providing an advanced starting solution for an exact algorithm.

END
7-87
DTIC